EZ-USB® FX3/FX3S SDK
Firmware API Guide

Version 1.3.1

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Contents

Chapter 1

EZ-USB FX3 Api Reference Guide

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

1.1 Introduction

Description

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

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

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

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

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

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

1.2 EZ-USB FX3 SDK

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

Description

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

1.2.1 FX3 API Libraries

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

Description

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

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

The following libraries are part of the SDK release and can be found under the firmware/u3p_firmware/lib folder in the installation:

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

The API definitions for all of these libraries is provided in the header files under the firmware/u3p_firmware/inc folder in the SDK installation.

1.2.2 FX3 Boot API Library

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

Description

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

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

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

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

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

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

1.2.3 FX3 Application Examples

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

Description

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

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

1.3 Embedded Real Time Operating System

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

Description

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

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

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

See Also

CyU3PThread
CyU3PQueue
CyU3PMutex
CyU3PSemaphore
CyU3PEvent
CyU3PTimer
CyU3PBytePool
CyU3PBlockPool

1.4 DMA Management

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

Description

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

1.4.1 DMA Sockets

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

Description

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

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

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

See Also

CyU3PDmaSocket_t

1.4.2 DMA Buffers

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

Description

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

1.4.3 DMA Descriptors

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

Description

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

See Also

CyU3PDmaDescriptor_t

1.4.4 DMA Channels

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

Description

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

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

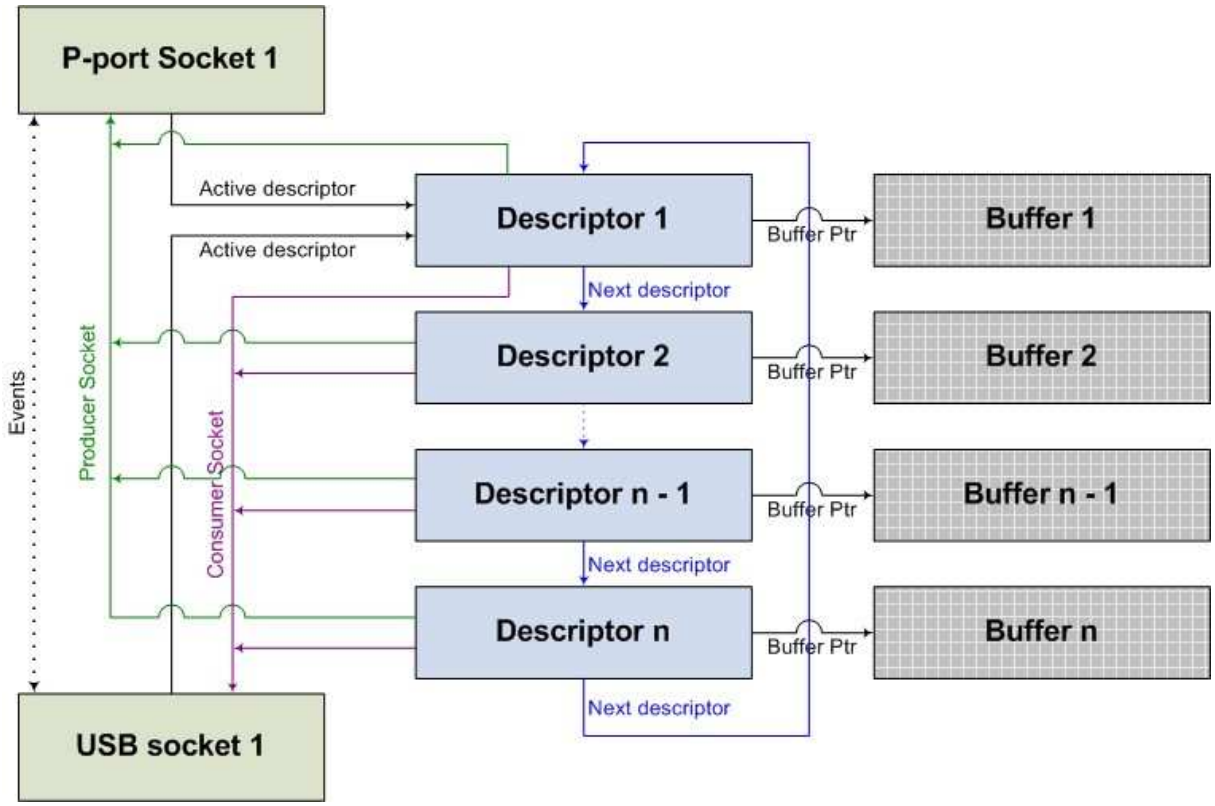

Figure 1.1: Resource linkage in a DMA channel

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

The following channel types are supported:

- Simple Channels

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

  1. Auto Channel

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

  2. Auto Channel with Signalling

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

  3. Manual Channel

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

  4. Manual IN Channel

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

5. Manual OUT Channel

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

- Multi-Channels

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

  1. One-to-Many Auto Channel

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

  2. One-to-Many Manual Channel

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

  3. Many-to-One Auto Channel

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

  4. Many-to-One Manual Channel

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

  5. Multicast Channel

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

See Also

CyU3PDmaType_t
CyU3PDmaChannel
CyU3PDmaMultiType_t
CyU3PDmaMultiChannel

1.5 Logging Support

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

Description

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

The log messages are classified into two types:

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

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

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

See Also

CyU3PDebugInit
CyU3PDebugPrint
CyU3PDebugLog

1.6 USB Driver and API

The USB driver and APIs allow users to configure the USB functionality of the FX3 device in peripheral as well as OTG/Host mode of operation.

Description

The USB block on the FX3 device is capable of functioning as a USB 3.0 peripheral, a USB 2.0 peripheral or a USB 2.0 host. It also supports USB 2.0 OTG functionality and Battery Charger detection.

The USB driver takes care of the device operation in each of these modes, and handles data transfers in a USB specification compliant manner. The USB APIs allow the user to configure the USB block and the driver in the appropriate mode as well as configure the device endpoints as required.

1.6.1 USB Peripheral (Device) Mode

The most common operating mode of the USB block is that of a peripheral device. The USB API allows the user to customize the USB peripheral functionality as required.

Description

The USB driver and API control the operation of the FX3 device as a USB peripheral in USB SuperSpeed, Hi-Speed and Full Speed modes. The driver detects the connection speed and takes care of providing the appropriate USB descriptors as well as setting the endpoint parameters as required. This ensures that a fully USB specification compliant application can be implemented by following the few implementation guidelines.

In the default operating mode, the FX3 detects the voltage on the VBus input and attempts to connect to the host only when a valid VBus voltage is detected. It also disables the USB connection and turns off the PHY block when the VBus voltage is removed.

See Also

CyU3PUsbStart
CyU3PConnectState

1.6.2 USB 2.0 Host Mode

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

Description

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

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

See Also

CyU3PUsbHostStart

1.6.3 USB On-The-Go (OTG) Mode

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

Description

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

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

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

See Also

CyU3POtgMode_t
CyU3POtgStart

1.7 Serial Peripheral Interfaces

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

Description

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

1. UART
2. I2C
3. SPI
4. I2S

5. GPIOs

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

1.7.1 UART Interface

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

Description

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

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

See Also

CyU3PUartInit
CyU3PUartSetConfig

1.7.2 I2C interface

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

Description

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

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

See Also

CyU3PI2cInit
CyU3PI2cSetConfig

1.7.3 SPI interface

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

Description

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

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

See Also

CyU3PSpiInit
CyU3PSpiSetConfig

1.7.4 I2S interface

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

Description

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

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

See Also

CyU3PI2sInit
CyU3PI2sSetConfig

1.7.5 General Purpose IO (GPIO) Support

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

Description

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

The FX3 device supports two classes of GPIO pins:

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

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

See Also

CyU3PGpioInit
CyU3PGpioSetSimpleConfig
CyU3PGpioSetComplexConfig

1.8 GPIF II Driver and API

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

Description

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

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

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

1.8.1 GPIF Configuration

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

Description

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

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

See Also

CyU3PGpifLoad
CyU3PGpifWaveformLoad
CyU3PGpifInitTransFunctions
CyU3PGpifConfigure

1.8.2 GPIF-II Resources

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

Description

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

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

There are three comparators: a control comparator, an address comparator and a data comparator that can be used to continuously check the current values on the control, address and data signal groups against a user configured pattern. The patterns for comparison can be programmed initially through the CyU3PGpifLoad() API and at later points through the CyU3PGpifInitComparator() API. When any of the comparators detect a match, a GPIF callback function can be generated. The comparator match can also be used as a transition trigger variable in the state machine.

See Also

CyU3PGpifInitComparator
CyU3PGpifInitCtrlCounter
CyU3PGpifInitDataCounter
CyU3PGpifInitAddrCounter

1.8.3 GPIF State Machine Control

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

Description

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

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

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

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

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

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

See Also

CyU3PGpifSMStart
CyU3PGpifSMSwitch
CyU3PGpifDisable

1.9 Storage Driver and API

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

Description

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

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

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

See Also

CyU3PSibStart
CyU3PSibQueryDevice
CyU3PSibReadWriteRequest
CyU3PSibVendorAccess
CyU3PSdioByteReadWrite
CyU3PSdioExtendedReadWrite

1.10 MIPI-CSI2 and Fixed Function GPIF Interface for CX3

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

Description

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

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

Additionally, example projects which demonstrate the usage of CX3 MIPI-CSI2 APIs are provided. The example projects implement high-definition (1920 x 1080 pixels, 16/24Bits per pixel) uncompressed video transfers at upto 30 frames per second from an Aptina AS0260 image sensor over USB 3.0. An example project demonstrating 5 Megapixel (2592 x 1944 pixels, 16 bits per pixel) uncompressed video transfers at upto 15 frames per second from an Omnivision OV5640 image sensor over USB 3.0 is also provided.

See Also

CyU3PMipicsiInit
CyU3PMipicsiDeInit
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams
CyU3PMipicsiReset
CyU3PMipicsiSleep
CyU3PMipicsiWakeup
CyU3PMipicsiGetErrors
CyU3PMipicsiGpifLoad

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 CyFx3BootGpioSimpleConfig_t Struct Reference

Configuration information for simple GPIOs.

`#include <cyfx3gpio.h>`

Data Fields

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

4.1.1 Detailed Description

Configuration information for simple GPIOs.

Description

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

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

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

See Also

CyFx3BootGpioIntrMode_t

4.1.2 Field Documentation

4.1.2.1 CyBool_t driveHighEn

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

otherwise tristated.

4.1.2.2 CyBool_t driveLowEn

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

otherwise tristated.

4.1.2.3 CyBool_t inputEn

When set true, the input state is enabled.

4.1.2.4 CyFx3BootGpioIntrMode_t intrMode

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

4.1.2.5 CyBool_t outValue

```
Initial value of the GPIO if configured as output:
```

CyFalse = 0, CyTrue = 1.

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

- firmware/boot_fw/include/cyfx3gpio.h

4.2 CyFx3BootI2cConfig_t Struct Reference

Structure defining the configuration of the I2C interface.

`#include <cyfx3i2c.h>`

Data Fields

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

4.2.1 Detailed Description

Structure defining the configuration of the I2C interface.

Description

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

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

See Also

CyFx3BootI2cSetConfig

4.2.2 Field Documentation

4.2.2.1 uint32_t bitRate

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

4.2.2.2 uint32_t busTimeout

```
Number of core clocks that SCK can be held low for by the slave byte
```

transmission before triggering a timeout error. 0xFFFFFFFFU means no timeout.

4.2.2.3 uint16_t dmaTimeout

```
Number of core clocks DMA can remain not ready before flagging an error.
```

0xFFFF means no timeout.

4.2.2.4 CyBool_t isDma

CyFalse: Register transfer mode, CyTrue: DMA transfer mode

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

- firmware/boot_fw/include/cyfx3i2c.h

4.3 CyFx3BootI2cPreamble_t Struct Reference

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

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

Data Fields

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

4.3.1 Detailed Description

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

Description

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

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

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

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

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

Byte 3:

Bit 7 - 1: Slave address.

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

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

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

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

See Also

CyFx3BootI2cSetConfig

4.3.2 Field Documentation

4.3.2.1 uint8_t buffer[8]

The extended preamble information.

4.3.2.2 uint16_t ctrlMask

```
This field controls the start stop condition after every byte of
```

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

4.3.2.3 uint8_t length

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

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

- firmware/boot_fw/include/cyfx3i2c.h

4.4 CyFx3BootIoMatrixConfig_t Struct Reference

Defines the IO matrix configuration parameters.

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

Data Fields

- CyBool_t isDQ32Bit
- CyBool_t useUart
- CyBool_t useI2C
- CyBool_t useI2S
- CyBool_t useSpi
- uint32_t gpioSimpleEn [2]

4.4.1 Detailed Description

Defines the IO matrix configuration parameters.

Description

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

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

See Also

CyFx3BootDeviceConfigureIOMatrix
CyFx3BootGpioOverride

4.4.2 Field Documentation

4.4.2.1 uint32_t gpioSimpleEn[2]

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

4.4.2.2 CyBool_t isDQ32Bit

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

4.4.2.3 CyBool_t useI2C

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

be used

4.4.2.4 CyBool_t useI2S

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

be used.

4.4.2.5 CyBool_t useSpi

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

be used.

4.4.2.6 CyBool_t useUart

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

to be used

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

- firmware/boot_fw/include/cyfx3device.h

4.5 CyFx3BootSpiConfig_t Struct Reference

Structure defining the configuration of SPI interface.

`#include <cyfx3spi.h>`

Data Fields

- CyBool_t isLsbFirst
- CyBool_t cpol
- CyBool_t cpha
- CyBool_t ssnPol
- CyFx3BootSpiSsnCtrl_t ssnCtrl
- CyFx3BootSpiSsnLagLead_t leadTime
- CyFx3BootSpiSsnLagLead_t lagTime
- uint32_t clock
- uint8_t wordLen

4.5.1 Detailed Description

Structure defining the configuration of SPI interface.

Description

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

See Also

CyFx3BootSpiSetConfig
CyFx3BootSpiSsnCtrl_t
CyFx3BootSpiSsnLagLead_t

4.5.2 Field Documentation

4.5.2.1 uint32_t clock

SPI interface clock frequency in Hz.

4.5.2.2 CyBool_t cpha

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

CyTrue: Slave samples at active-idle edge

4.5.2.3 CyBool_t cpol

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

4.5.2.4 CyBool_t isLsbFirst

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

4.5.2.5 CyFx3BootSpiSsnLagLead_t lagTime

```
Time between the last SCK edge to SSN de-assertion. This is at
```

the end of a transfer and is valid only for hardware controlled SSN.

4.5.2.6 CyFx3BootSpiSsnLagLead_t leadTime

```
Time between SSN assertion and first SCLK edge. This is at the
```

beginning of a transfer and is valid only for hardware controlled SSN. Zero lead time is not supported.

4.5.2.7 CyFx3BootSpiSsnCtrl_t ssnCtrl

Mode of SSN control.

4.5.2.8 CyBool_t ssnPol

```
Polarity of SSN line. CyFalse: SSN is active low;
```

CyTrue: SSN is active high.

4.5.2.9 uint8_t wordLen

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

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

- firmware/boot_fw/include/cyfx3spi.h

4.6 CyFx3BootUartConfig_t Struct Reference

Configuration parameters for the UART interface.

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

Data Fields

- CyBool_t txEnable
- CyBool_t rxEnable
- CyBool_t flowCtrl
- CyBool_t isDma
- CyFx3BootUartBaudrate_t baudRate
- CyFx3BootUartStopBit_t stopBit
- CyFx3BootUartParity_t parity

4.6.1 Detailed Description

Configuration parameters for the UART interface.

Description

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

See Also

CyFx3BootUartBaudrate_t
CyFx3BootUartStopBit_t
CyFx3BootUartParity_t
CyFx3BootUartSetConfig

4.6.2 Field Documentation

4.6.2.1 CyFx3BootUartBaudrate_t baudRate

Baud rate for data transfer.

4.6.2.2 CyBool_t flowCtrl

Enable Flow control for both RX and TX.

4.6.2.3 CyBool_t isDma

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

4.6.2.4 CyFx3BootUartParity_t parity

Parity configuration.

4.6.2.5 CyBool_t rxEnable

Enable the receiver.

4.6.2.6 CyFx3BootUartStopBit_t stopBit

The number of stop bits used.

4.6.2.7 CyBool_t txEnable

Enable the transmitter.

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

- firmware/boot_fw/include/cyfx3uart.h

4.7 CyFx3BootUsbEp0Pkt_t Struct Reference

USB Setup Packet data structure.

`#include <cyfx3usb.h>`

Data Fields

- uint8_t bmReqType
- uint8_t bReq
- uint8_t bVal0
- uint8_t bVal1
- uint8_t bIdx0
- uint8_t bIdx1
- uint16_t wLen
- uint8_t * pData

4.7.1 Detailed Description

USB Setup Packet data structure.

Description

Control Endpoint setup packet data structure.

4.7.2 Field Documentation

4.7.2.1 uint8_t bIdx0

wIndex field, LSB.

4.7.2.2 uint8_t bIdx1

wIndex field, MSB.

4.7.2.3 uint8_t bmReqType

Direction, type of request and intended recipient

4.7.2.4 uint8_t bReq

Request being made

4.7.2.5 uint8_t bVal0

wValue field, LSB.

4.7.2.6 uint8_t bVal1

wValue field, MSB.

4.7.2.7 uint8_t* pData

Pointer to the data

4.7.2.8 uint16_t wLen

wLength field.

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

- firmware/boot_fw/include/cyfx3usb.h

4.8 CyFx3BootUsbEpConfig_t Struct Reference

Endpoint configuration information.

`#include <cyfx3usb.h>`

Data Fields

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

4.8.1 Detailed Description

Endpoint configuration information.

Description

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

4.8.2 Field Documentation

4.8.2.1 uint8_t burstLen

Maximum burst length in packets.

4.8.2.2 CyBool_t enable

Endpoint status - enabled or not.

4.8.2.3 CyFx3BootUsbEpType_t epType

The endpoint type

4.8.2.4 uint8_t isoPkts

Number of packets per micro-frame for ISO endpoints.

4.8.2.5 uint16_t pcktSize

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

4.8.2.6 uint16_t streams

Number of bulk streams used by the endpoint.

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

- firmware/boot_fw/include/cyfx3usb.h

4.9 CyU3PDebugLog_t Struct Reference

FX3 debug logger data type.

`#include <cyu3system.h>`

Data Fields

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

4.9.1 Detailed Description

FX3 debug logger data type.

Description

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

See Also

CyU3PDebugLog
CyU3PDebugPrint

4.9.2 Field Documentation

4.9.2.1 uint16_t msg

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

4.9.2.2 uint32_t param

32 bit message parameter.

4.9.2.3 uint8_t priority

Priority of the message

4.9.2.4 uint8_t threadId

Id of thread sending the message.

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

- firmware/u3p_firmware/inc/cyu3system.h

4.10 CyU3PDmaBuffer_t Struct Reference

DMA buffer data structure.

`#include <cyu3dma.h>`

Data Fields

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

4.10.1 Detailed Description

DMA buffer data structure.

Description

The data structure is used to describe the status of a DMA buffer. It holds the address, size, valid data count, and the status information. This type is used in DMA callbacks and APIs to identify the properties of the data buffer to be transferred.

See Also

CyU3PDmaCBInput_t
CyU3PDmaChannelGetBuffer
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForRecvBuffer
CyU3PDmaMultiChannelGetBuffer
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForRecvBuffer

4.10.2 Field Documentation

4.10.2.1 uint8_t* buffer

Pointer to the data buffer.

4.10.2.2 uint16_t count

Byte count of valid data in buffer.

4.10.2.3 uint16_t size

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

4.10.2.4 uint16_t status

```
Buffer status. This is a four bit data field defined by
```

CY_U3P_DMA_BUFFER_STATUS_MASK. This holds information like whether the buffer is occupied, whether the buffer holds the end of packet and whether the buffer encountered a DMA error.

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.11 CyU3PDmaCBInput_t Union Reference

DMA channel callback input.

`#include <cyu3dma.h>`

Data Fields

- CyU3PDmaBuffer_t buffer_p

4.11.1 Detailed Description

DMA channel callback input.

Description

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

See Also

CyU3PDmaBuffer_t
CyU3PDmaCallback_t

4.11.2 Field Documentation

4.11.2.1 CyU3PDmaBuffer_t buffer_p

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

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.12 CyU3PDmaChannel Struct Reference

DMA Channel structure.

`#include <cyu3dma.h>`

Data Fields

- 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

4.12.1 Detailed Description

DMA Channel structure.

Description

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

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaChannelDestroy
CyU3PDmaChannelUpdateMode
CyU3PDmaChannelGetStatus
CyU3PDmaChannelGetHandle
CyU3PDmaChannelSetXfer
CyU3PDmaChannelGetBuffer
CyU3PDmaChannelCommitBuffer
CyU3PDmaChannelDiscardBuffer
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForCompletion
CyU3PDmaChannelWaitForRecvBuffer
CyU3PDmaChannelSetWrapUp
CyU3PDmaChannelSetSuspend

CyU3PDmaChannelResume
CyU3PDmaChannelAbort
CyU3PDmaChannelReset
CyU3PDmaChannelCacheControl

4.12.2 Field Documentation

4.12.2.1 uint16_t activeConsIndex

Active consumer descriptor.

4.12.2.2 uint16_t activeProdIndex

Active producer descriptor.

4.12.2.3 CyU3PDmaCallback_t cb

Callback function which gets invoked on DMA events

4.12.2.4 uint16_t commitConsIndex

Consumer descriptor for the buffer to be consumed.

4.12.2.5 uint16_t commitProdIndex

Producer descriptor for the buffer to be consumed.

4.12.2.6 uint16_t consHeader

The consumer socket header offset

4.12.2.7 uint16_t consSckId

The consumer (egress) socket ID

4.12.2.8 uint16_t consSusp

Consumer suspend option.

4.12.2.9 uint16_t count

Number of buffers for the channel

4.12.2.10 uint16_t currentConsIndex

Consumer descriptor for the latest buffer produced.

4.12.2.11 uint16_t currentProdIndex

Producer descriptor for the latest buffer produced.

4.12.2.12 uint16_t discardCount

Number of pending discards.

4.12.2.13 uint16_t dmaMode

Mode of DMA operation

4.12.2.14 uint16_t firstConsIndex

Head for the manual mode consumer descriptor chain list

4.12.2.15 uint16_t firstProdIndex

Head for the normal descriptor chain list

4.12.2.16 CyU3PEvent flags

Event flags for the channel

4.12.2.17 CyBool_t isDmaHandleDCache

Whether to do DMA cache handling for this channel.

4.12.2.18 CyU3PMutex lock

Lock for this channel structure.

4.12.2.19 uint32_t notification

Registered notifications

4.12.2.20 uint16_t overrideDscrIndex

Descriptor for override modes.

4.12.2.21 uint16_t prodAvailCount

Minimum available buffers before producer is active.

4.12.2.22 uint16_t prodFooter

The producer socket footer offset

4.12.2.23 uint16_t prodHeader

The producer socket header offset

4.12.2.24 uint16_t prodSckId

The producer (ingress) socket ID

4.12.2.25 uint16_t prodSusp

Producer suspend option.

4.12.2.26 uint16_t size

The buffer size associated with the channel

4.12.2.27 uint32_t state

The current state of the DMA channel

4.12.2.28 uint16_t type

The type of the DMA channel

4.12.2.29 uint32_t xferSize

Current transfer size

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.13 CyU3PDmaChannelConfig_t Struct Reference

DMA channel parameters.

`#include <cyu3dma.h>`

Data Fields

- uint16_t size
- uint16_t count
- CyU3PDmaSocketId_t prodSckId
- CyU3PDmaSocketId_t consSckId
- uint16_t prodAvailCount
- uint16_t prodHeader
- uint16_t prodFooter
- uint16_t consHeader
- CyU3PDmaMode_t dmaMode
- uint32_t notification
- CyU3PDmaCallback_t cb

4.13.1 Detailed Description

DMA channel parameters.

Description

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

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

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

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

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

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

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

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

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

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

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

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

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

See Also

CyU3PDmaChannelCreate

4.13.2 Field Documentation

4.13.2.1 CyU3PDmaCallback_t cb

Callback function which gets invoked on DMA events.

4.13.2.2 uint16_t consHeader

The consumer socket header offset.

4.13.2.3 CyU3PDmaSocketId_t consSckId

The consumer (egress) socket ID.

4.13.2.4 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.

4.13.2.5 CyU3PDmaMode_t dmaMode

Mode of DMA operation.

4.13.2.6 uint32_t notification

```
Registered notifications. This is a bit map based on CyU3PDmaCbType_t.
```

This defines the events for which the callback is triggered.

4.13.2.7 uint16_t prodAvailCount

```
Minimum available empty buffers before producer is active. By default,
```

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

4.13.2.8 uint16_t prodFooter

The producer socket footer offset.

4.13.2.9 uint16_t prodHeader

The producer socket header offset.

4.13.2.10 CyU3PDmaSocketId_t prodSckId

The producer (ingress) socket ID.

4.13.2.11 uint16_t size

The buffer size associated with the channel.

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.14 CyU3PDmaDescriptor_t Struct Reference

Descriptor data structure.

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

Data Fields

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

4.14.1 Detailed Description

Descriptor data structure.

Description

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

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

buffer: (CY_U3P_BUFFER_ADDR_MASK)

sync: (CY_U3P_EN_PROD_INT | CY_U3P_EN_PROD_EVENT | CY_U3P_PROD_IP_MASK | CY_U3P_PROD_SCK_MASK | CY_U3P_EN_CONS_INT | CY_U3P_EN_CONS_EVENT | CY_U3P_CONS_IP_MASK | CY_U3P_CONS_SCK_MASK)

chain: (CY_U3P_WR_NEXT_DSCR_MASK | CY_U3P_RD_NEXT_DSCR_MASK)

size: (CY_U3P_BYTE_COUNT_MASK | CY_U3P_BUFFER_SIZE_MASK | CY_U3P_BUFFER_OCCUPIED | CY_U3P_BUFFER_ERROR | CY_U3P_EOP | CY_U3P_MARKER)

See Also

CyU3PDmaDscrGetConfig
CyU3PDmaDscrSetConfig

4.14.2 Field Documentation

4.14.2.1 uint8_t* buffer

Pointer to buffer used.

4.14.2.2 uint32_t chain

Next descriptor links.

4.14.2.3 uint32_t size

Current and maximum sizes of buffer.

4.14.2.4 uint32_t sync

Consumer, Producer binding.

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

- firmware/u3p_firmware/inc/cyu3descriptor.h

4.15 CyU3PDmaMultiChannel Struct Reference

DMA multi-channel structure.

`#include <cyu3dma.h>`

Data Fields

- uint32_t state
- uint16_t type
- uint16_t size
- uint16_t count
- uint16_t validSckCount
- uint16_t consDisabled [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t firstProdIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t firstConsIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t prodSckId [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t consSckId [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t overrideDscrIndex
- uint16_t currentProdIndex
- uint16_t currentConsIndex
- uint16_t commitProdIndex
- uint16_t commitConsIndex
- uint16_t activeProdIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t activeConsIndex [CY_U3P_DMA_MAX_MULTI_SCK_COUNT]
- uint16_t prodHeader
- uint16_t prodFooter
- uint16_t consHeader
- uint16_t prodAvailCount
- uint16_t dmaMode
- uint16_t prodSusp
- uint16_t consSusp
- uint16_t 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

4.15.1 Detailed Description

DMA multi-channel structure.

Description

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

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelConfig_t
CyU3PDmaMultiChannelCreate
CyU3PDmaMultiChannelDestroy
CyU3PDmaMultiChannelGetHandle
CyU3PDmaMultiChannelUpdateMode
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelGetBuffer
CyU3PDmaMultiChannelCommitBuffer
CyU3PDmaMultiChannelDiscardBuffer
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForCompletion
CyU3PDmaMultiChannelWaitForRecvBuffer
CyU3PDmaMultiChannelSetWrapUp
CyU3PDmaMultiChannelSetSuspend
CyU3PDmaMultiChannelResume
CyU3PDmaMultiChannelAbort
CyU3PDmaMultiChannelReset
CyU3PDmaMultiChannelGetStatus
CyU3PDmaMultiChannelCacheControl

4.15.2 Field Documentation

4.15.2.1 uint16_t activeConsIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Active consumer descriptor.

4.15.2.2 uint16_t activeProdIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Active producer descriptor.

4.15.2.3 uint16_t bufferCount[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Number of active buffers available for consumer.

4.15.2.4 CyU3PDmaMultiCallback_t cb

Callback function which gets invoked on DMA events

4.15.2.5 uint16_t commitConsIndex

Consumer descriptor for the buffer to be consumed.

4.15.2.6 uint16_t commitProdIndex

Producer descriptor for the buffer to be consumed.

4.15.2.7 uint16_t consDisabled[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

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

4.15.2.8 uint16_t consHeader

The consumer socket header offset

4.15.2.9 uint16_t consSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

The consumer (egress) socket ID

4.15.2.10 uint16_t consSusp

Consumer suspend option.

4.15.2.11 uint16_t count

Number of buffers for the channel

4.15.2.12 uint16_t currentConsIndex

Consumer descriptor for the latest buffer produced.

4.15.2.13 uint16_t currentProdIndex

Producer descriptor for the latest buffer produced.

4.15.2.14 uint16_t discardCount[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Number of buffers to be discarded.

4.15.2.15 uint16_t dmaMode

Mode of DMA operation

4.15.2.16 uint16_t firstConsIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Head for the manual consumer descriptor chain

4.15.2.17 uint16_t firstProdIndex[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

Head for the normal descriptor chain

4.15.2.18 CyU3PEvent flags

Event flags for the channel

4.15.2.19 CyBool_t isDmaHandleDCache

Whether to do internal DMA cache handling.

4.15.2.20 CyU3PMutex lock

Lock for the channel structure

4.15.2.21 uint32_t notification

Registered notifications.

4.15.2.22 uint16_t overrideDscrIndex

Descriptor for override modes.

4.15.2.23 uint16_t prodAvailCount

Minimum available buffers before producer is active.

4.15.2.24 uint16_t prodFooter

The producer socket footer offset

4.15.2.25 uint16_t prodHeader

The producer socket header offset

4.15.2.26 uint16_t prodSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

The producer (ingress) socket ID

4.15.2.27 uint16_t prodSusp

Producer suspend option.

4.15.2.28 uint16_t size

The buffer size associated with the channel

4.15.2.29 uint32_t state

The current state of the DMA channel

4.15.2.30 uint16_t type

The type of the DMA channel

4.15.2.31 uint16_t validSckCount

Number of sockets in the multi-socket operation.

4.15.2.32 uint32_t xferSize

Current xfer size

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.16 CyU3PDmaMultiChannelConfig_t Struct Reference

DMA multi-channel parameter structure.

`#include <cyu3dma.h>`

Data Fields

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

4.16.1 Detailed Description

DMA multi-channel parameter structure.

Description

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

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

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

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

The size, count, prodHeader, prodFooter, consHeader and prodAvailCount fields are used in the same way as in the CyU3PDmaChannelConfig_t structure.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelCreate

4.16.2 Field Documentation

4.16.2.1 CyU3PDmaMultiCallback_t cb

Callback function which gets invoked on multi socket DMA events.

4.16.2.2 uint16_t consHeader

The consumer socket header offset.

4.16.2.3 CyU3PDmaSocketId_t consSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

The consumer (egress) socket ID.

4.16.2.4 uint16_t count

```
Number of buffers to be allocated for each socket of the channel.
```

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

4.16.2.5 CyU3PDmaMode_t dmaMode

Mode of DMA operation

4.16.2.6 uint32_t notification

```
Registered notifications. This is a bit map based on CyU3PDmaCbType_t.
```

This defines the events for which the callback is triggered.

4.16.2.7 uint16_t prodAvailCount

```
Minimum available empty buffers before producer is active. By default,
```

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

4.16.2.8 uint16_t prodFooter

The producer socket footer offset.

4.16.2.9 uint16_t prodHeader

The producer socket header offset.

4.16.2.10 CyU3PDmaSocketId_t prodSckId[CY_U3P_DMA_MAX_MULTI_SCK_COUNT]

The producer (ingress) socket ID.

4.16.2.11 uint16_t size

The buffer size associated with the channel.

4.16.2.12 uint16_t validSckCount

Number of sockets in the multi-socket operation.

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

- firmware/u3p_firmware/inc/cyu3dma.h

4.17 CyU3PDmaSocket_t Struct Reference

DMA socket register structure.

`#include <cyu3socket.h>`

Data Fields

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

4.17.1 Detailed Description

DMA socket register structure.

Description

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

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

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

See Also

CyU3PDmaSocketConfig_t

4.17.2 Field Documentation

4.17.2.1 CyU3PDmaDescriptor_t activeDscr

Active descriptor information. See cyu3descriptor.h for definition.

4.17.2.2 uvint32_t dscrChain

The descriptor chain associated with the socket

4.17.2.3 uvint32_t intr

Interrupt status register.

4.17.2.4 uvint32_t intrMask

Interrupt mask register.

4.17.2.5 uvint32_t sckEvent

Generate event register.

4.17.2.6 uvint32_t status

Socket configuration and status register.

4.17.2.7 uvint32_t unused19[19]

Reserved register space.

4.17.2.8 uvint32_t unused2[2]

Reserved register space.

4.17.2.9 uvint32_t xferCount

The completed transfer count for this socket.

4.17.2.10 uvint32_t xferSize

```
The transfer size requested for this socket. The size can
```

be specified in bytes or in terms of number of buffers, depending on the UNIT field in the status value.

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

- firmware/u3p_firmware/inc/cyu3socket.h

4.18 CyU3PDmaSocketConfig_t Struct Reference

DMA socket configuration structure.

`#include <cyu3socket.h>`

Data Fields

- uint32_t dscrChain
- uint32_t xferSize
- uint32_t xferCount
- uint32_t status
- uint32_t intr
- uint32_t intrMask

4.18.1 Detailed Description

DMA socket configuration structure.

Description

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

See Also

CyU3PDmaSocket_t
CyU3PDmaSocketSetConfig
CyU3PDmaSocketGetConfig

4.18.2 Field Documentation

4.18.2.1 uint32_t dscrChain

The descriptor chain associated with the socket.

4.18.2.2 uint32_t intr

Interrupt status.

4.18.2.3 uint32_t intrMask

Interrupt mask.

4.18.2.4 uint32_t status

Socket status register.

4.18.2.5 uint32_t xferCount

Transfer status for the socket.

4.18.2.6 uint32_t xferSize

Transfer size for the socket.

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

- firmware/u3p_firmware/inc/cyu3socket.h

4.19 CyU3PEpConfig_t Struct Reference

Endpoint configuration information.

`#include <cyu3usb.h>`

Data Fields

- CyBool_t enable
- CyU3PUsbEpType_t epType
- uint16_t streams
- uint16_t pcktSize
- uint8_t burstLen
- uint8_t isoPkts

4.19.1 Detailed Description

Endpoint configuration information.

Description

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

See Also

CyU3PSetEpConfig

4.19.2 Field Documentation

4.19.2.1 uint8_t burstLen

```
Maximum burst length in packets. This needs to be specified as the
```

number of packets per burst and not as "burst length - 1" as in the case of the Super Speed Companion descriptor.

4.19.2.2 CyBool_t enable

Endpoint status - enabled or not.

4.19.2.3 CyU3PUsbEpType_t epType

```
The endpoint type - Isochronous = 1, Bulk = 2, Interrupt = 3.
```

See USB specification for type values.

4.19.2.4 uint8_t isoPkts

Number of packets per micro-frame for ISO endpoints.

4.19.2.5 uint16_t pcktSize

```
Maximum packet size for the endpoint. Can be set to 0 if the maximum
```

packet size should be set to the maximum value consistent with the USB specification.

4.19.2.6 uint16_t streams

```
Number of bulk streams used by the endpoint. This needs to be specified
```

as the number of streams and not as "stream count - 1" as in the case of the Super Speed Companion descriptor.

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

- firmware/u3p_firmware/inc/cyu3usb.h

4.20 CyU3PGpifConfig_t Struct Reference

Structure that holds all configuration inputs for the GPIF hardware.

`#include <cyu3gpif.h>`

Data Fields

- const uint16_t stateCount
- const CyU3PGpifWaveData * stateData
- const uint8_t * statePosition
- const uint16_t functionCount
- const uint16_t * functionData
- const uint16_t regCount
- const uint32_t * regData

4.20.1 Detailed Description

Structure that holds all configuration inputs for the GPIF hardware.

Description

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

See Also

CyU3PGpifLoad
CyU3PGpifWaveData

4.20.2 Field Documentation

4.20.2.1 const uint16_t functionCount

Number of transition functions to be initialized.

4.20.2.2 const uint16_t* functionData

Pointer to array containing transition function data.

4.20.2.3 const uint16_t regCount

Number of GPIF config registers to be initialized.

4.20.2.4 const uint32_t* regData

Pointer to array containing GPIF register values.

4.20.2.5 const uint16_t stateCount

Number of states to be initialized.

4.20.2.6 const CyU3PGpifWaveData* stateData

Pointer to array containing state descriptors.

4.20.2.7 const uint8_t* statePosition

Pointer to array index -> state number mapping.

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

- firmware/u3p_firmware/inc/cyu3gpif.h

4.21 CyU3PGpifWaveData Struct Reference

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

`#include <cyu3gpif.h>`

Data Fields

- uint32_t leftData [3]
- uint32_t rightData [3]

4.21.1 Detailed Description

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

Description

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

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

See Also

CyU3PGpifWaveformLoad

4.21.2 Field Documentation

4.21.2.1 uint32_t leftData[3]

12 byte left transition descriptor.

4.21.2.2 uint32_t rightData[3]

12 byte right transition descriptor.

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

- firmware/u3p_firmware/inc/cyu3gpif.h

4.22 CyU3PGpioClock_t Struct Reference

Clock configuration information for the GPIO block.

`#include <cyu3lpp.h>`

Data Fields

- uint8_t fastClkDiv
- uint8_t slowClkDiv
- CyBool_t halfDiv
- CyU3PGpioSimpleClkDiv_t simpleDiv
- CyU3PSysClockSrc_t clkSrc

4.22.1 Detailed Description

Clock configuration information for the GPIO block.

Description

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

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

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

See Also

CyU3PGpioSetClock
CyU3PSysClockSrc_t
CyU3PGpioSimpleClkDiv_t

4.22.2 Field Documentation

4.22.2.1 CyU3PSysClockSrc_t clkSrc

The clock source to be used for this peripheral.

4.22.2.2 uint8_t fastClkDiv

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

value is 16.

4.22.2.3 CyBool_t halfDiv

```
This allows the fast clock to be divided by a non integral value.
```

If set to true, this adds 0.5 to the fastClkDiv value. This should be used only if the slow clock is disabled.

4.22.2.4 CyU3PGpioSimpleClkDiv_t simpleDiv

Divider value from fast clock for sampling simple GPIOs.

4.22.2.5 uint8_t slowClkDiv

```
Divider value for the GPIO slow clock. This divisor applies on top of
```

the fast clock and can be set to zero, to disable the slow clock. The min value is 0 (1 is not supported) and max value is 64.

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

- firmware/u3p_firmware/inc/cyu3lpp.h

4.23 CyU3PGpioComplexConfig_t Struct Reference

Configuration information for complex GPIO pins.

`#include <cyu3gpio.h>`

Data Fields

- CyBool_t outValue
- CyBool_t driveLowEn
- CyBool_t driveHighEn
- CyBool_t inputEn
- CyU3PGpioComplexMode_t pinMode
- CyU3PGpioIntrMode_t intrMode
- CyU3PGpioTimerMode_t timerMode
- uint32_t timer
- uint32_t period
- uint32_t threshold

4.23.1 Detailed Description

Configuration information for complex GPIO pins.

Description

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

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

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioSimpleConfig_t
CyU3PGpioComplexMode_t
CyU3PGpioTimerMode_t
CyU3PGpioIntrMode_t

4.23.2 Field Documentation

4.23.2.1 CyBool_t driveHighEn

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

otherwise tristated.

4.23.2.2 CyBool_t driveLowEn

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

otherwise tristated.

4.23.2.3 CyBool_t inputEn

When set true, the input state is enabled.

4.23.2.4 CyU3PGpioIntrMode_t intrMode

Interrupt mode for the GPIO.

4.23.2.5 CyBool_t outValue

```
Initial value for the GPIO if configured as output:
```

CyFalse = 0, CyTrue = 1.

4.23.2.6 uint32_t period

Timer period.

4.23.2.7 CyU3PGpioComplexMode_t pinMode

Complex GPIO operating mode.

4.23.2.8 uint32_t threshold

Timer threshold value.

4.23.2.9 uint32_t timer

Timer initial value.

4.23.2.10 CyU3PGpioTimerMode_t timerMode

Timer mode.

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

- firmware/u3p_firmware/inc/cyu3gpio.h

4.24 CyU3PGpioSimpleConfig_t Struct Reference

Configuration information for simple GPIOs.

`#include <cyu3gpio.h>`

Data Fields

- CyBool_t outValue
- CyBool_t driveLowEn
- CyBool_t driveHighEn
- CyBool_t inputEn
- CyU3PGpioIntrMode_t intrMode

4.24.1 Detailed Description

Configuration information for simple GPIOs.

Description

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

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

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

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

See Also

CyU3PGpioIntrMode_t
CyU3PGpioSetSimpleConfig

4.24.2 Field Documentation

4.24.2.1 CyBool_t driveHighEn

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

otherwise tristated.

4.24.2.2 CyBool_t driveLowEn

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

otherwise tristated.

4.24.2.3 CyBool_t inputEn

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

4.24.2.4 CyU3PGpioIntrMode_t intrMode

Interrupt mode for the GPIO.

4.24.2.5 CyBool_t outValue

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

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

- firmware/u3p_firmware/inc/cyu3gpio.h

4.25 CyU3PI2cConfig_t Struct Reference

Structure defining the I2C interface configuration.

`#include <cyu3i2c.h>`

Data Fields

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

4.25.1 Detailed Description

Structure defining the I2C interface configuration.

Description

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

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

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

See Also

CyU3PI2cSetConfig

4.25.2 Field Documentation

4.25.2.1 uint32_t bitRate

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

4.25.2.2 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.

4.25.2.3 uint16_t dmaTimeout

```
Number of core clocks DMA can remain not ready before flagging an error.
```

0xFFFF means no timeout.

4.25.2.4 CyBool_t isDma

CyFalse: Register transfer mode, CyTrue: DMA transfer mode

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

- firmware/u3p_firmware/inc/cyu3i2c.h

4.26 CyU3PI2cPreamble_t Struct Reference

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

`#include <cyu3i2c.h>`

Data Fields

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

4.26.1 Detailed Description

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

Description

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

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

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

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

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

Byte 3:

Bit 7 - 1: Slave address.

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

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

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

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

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

See Also

CyU3PI2cTransmitBytes
CyU3PI2cReceiveBytes

4.26.2 Field Documentation

4.26.2.1 uint8_t buffer[8]

The actual preamble bytes starting with slave address.

4.26.2.2 uint16_t ctrlMask

```
This field controls the start stop condition after every byte of
```

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

4.26.2.3 uint8_t length

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

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

- firmware/u3p_firmware/inc/cyu3i2c.h

4.27 CyU3PI2sConfig_t Struct Reference

I2S interface configuration parameters.

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

Data Fields

- CyBool_t isMono
- CyBool_t isLsbFirst
- CyBool_t isDma
- CyU3PI2sPadMode_t padMode
- CyU3PI2sSampleRate_t sampleRate
- CyU3PI2sSampleWidth_t sampleWidth

4.27.1 Detailed Description

I2S interface configuration parameters.

Description

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

See Also

CyU3PI2sSetConfig

4.27.2 Field Documentation

4.27.2.1 CyBool_t isDma

CyTrue: DMA mode, CyFalse: Register mode

4.27.2.2 CyBool_t isLsbFirst

CyTrue: LSB First, CyFalse: MSB First

4.27.2.3 CyBool_t isMono

CyTrue: Mono, CyFalse: Stereo

4.27.2.4 CyU3PI2sPadMode_t padMode

Type of padding to be used

4.27.2.5 CyU3PI2sSampleRate_t sampleRate

Sample rate for this audio stream.

4.27.2.6 CyU3PI2sSampleWidth_t sampleWidth

Bit width for samples in this audio stream.

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

- firmware/u3p_firmware/inc/cyu3i2s.h

4.28 CyU3PIoMatrixConfig_t Struct Reference

IO matrix configuration parameters.

`#include <cyfx3_api.h>`

Data Fields

- CyBool_t isDQ32Bit
- CyBool_t useUart
- CyBool_t useI2C
- CyBool_t useI2S
- CyBool_t useSpi
- CyU3PSPortMode_t s0Mode
- CyU3PSPortMode_t s1Mode
- CyU3PIoMatrixLppMode_t lppMode
- uint32_t gpioSimpleEn [2]
- uint32_t gpioComplexEn [2]

4.28.1 Detailed Description

IO matrix configuration parameters.

Description

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

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

Note

A common structure including the storage port configuration is used for both FX3 and FX3S devices, in order to maintain a common API interface. The s0Mode and s1Mode fields should be set to CY_U3P_SPORT_INACTIVE when using the EZ-USB FX3 device.

See Also

CyU3PIoMatrixLppMode_t
CyU3PSPortMode_t
CyU3PDeviceConfigureIOMatrix

4.28.2 Field Documentation

4.28.2.1 uint32_t gpioComplexEn[2]

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

4.28.2.2 uint32_t gpioSimpleEn[2]

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

4.28.2.3 CyBool_t isDQ32Bit

Whether the GPIF data bus is 32 bits wide.

4.28.2.4 CyU3PIoMatrixLppMode_t lppMode

LPP IO configuration to be used.

4.28.2.5 CyU3PSPortMode_t s0Mode

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

4.28.2.6 CyU3PSPortMode_t s1Mode

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

4.28.2.7 CyBool_t useI2C

Whether pins are to be reserved for the I2C interface.

4.28.2.8 CyBool_t useI2S

Whether pins are to be reserved for the I2S interface.

4.28.2.9 CyBool_t useSpi

Whether pins are to be reserved for the SPI interface.

4.28.2.10 CyBool_t useUart

Whether pins are to be reserved for the UART interface.

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

- firmware/u3p_firmware/inc/cyfx3_api.h

4.29 CyU3PMbox Struct Reference

Structure that holds a packet of mailbox data.

`#include <cyu3mbox.h>`

Data Fields

- uint32_t w0
- uint32_t w1

4.29.1 Detailed Description

Structure that holds a packet of mailbox data.

Description

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

4.29.2 Field Documentation

4.29.2.1 uint32_t w0

Contents of the lower mailbox register.

4.29.2.2 uint32_t w1

Contents of the upper mailbox register.

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

- firmware/u3p_firmware/inc/cyu3mbox.h

4.30 CyU3PMipicsiCfg_t Struct Reference

Structure defining the MIPI-CSI block interface configuration.

`#include <cyu3mipicsi.h>`

Data Fields

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

4.30.1 Detailed Description

Structure defining the MIPI-CSI block interface configuration.

Description

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

Note

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

See Also

CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams
CyU3PMipicsiPllClkFrs_t
CyU3PMipicsiPllClkDiv_t

4.30.2 Field Documentation

4.30.2.1 CyU3PMipicsiPllClkDiv_t csiRxClkDiv

```
Clock divider for generating clock used for detecting
```

CSI Link LP<->HS transitions

4.30.2.2 CyU3PMipicsiDataFormat_t dataFormat

```
Data Format expected at the GPIF.
```

The image sensor will also need to be separately set to transmit in this format.

4.30.2.3 uint16_t fifoDelay

```
Threshold at which the output from the parallel output buffer on the
```

MIPI-CSI interface will be initiated. Range 0x000 - 0x1FF

4.30.2.4 uint16_t hResolution

```
Horizontal Resolution defined in number of pixels per line of
```

the video frame.

4.30.2.5 uint16_t mClkCtl

```
Settings used to generate MCLK output clock (to provide
```

Reference clock to the attached Image Sensor). The output clock is available on the CLKM_CX3 line mClkOutput = (PLL_Clock/mClkRefDiv)/ [(CY_U3P_GET_MSB(mClkCtl) + 1) + (CY_U3P_GET_LSB(mclkCtl) + 1)]

4.30.2.6 CyU3PMipicsiPllClkDiv_t mClkRefDiv

Clock divider used for generating the MCLK

4.30.2.7 uint8_t numDataLanes

```
Number of MIPI-CSI data lanes to use. Valid values
```

are 1,2,3 and 4. The number of lanes which can be used varies depending on the part being used and the Image Sensor being used. Please refer to the CX3 part datasheet and the Image Sensor datasheet for more details on the number of lanes available.

4.30.2.8 CyU3PMipicsiPllClkDiv_t parClkDiv

```
Clock divider used for generating the PCLK used for clocking the
```

fixed function GPIF interface

4.30.2.9 uint16_t pllFbd

Feedback Divider used for generating MIPI-CSI PLL_CLOCK from

the input REFCLK. Valid setting range:0x000-0x1FF

4.30.2.10 CyU3PMipicsiPllClkFrs_t pllFrs

Frequency Range Setting for the MIPI CSI PLL_CLOCK

4.30.2.11 uint8_t pllPrd

Input Divider used for generating MIPI-CSI PLL_CLOCK from

the input REFCLK. Valid setting range: 0x0-0xF

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

- firmware/u3p_firmware/inc/cyu3mipicsi.h

4.31 CyU3PMipicsiErrorCounts_t Struct Reference

Structure defining MIPI-CSI block Phy errors.

`#include <cyu3mipicsi.h>`

Data Fields

- uint8_t frmErrCnt
- uint8_t crcErrCnt
- uint8_t mdlErrCnt
- uint8_t ctlErrCnt
- uint8_t eidErrCnt
- uint8_t recrErrCnt
- uint8_t unrcErrCnt
- uint8_t recSyncErrCnt
- uint8_t unrSyncErrCnt

4.31.1 Detailed Description

Structure defining MIPI-CSI block Phy errors.

Description

The following structure is used to fetch count of MIPI-CSI Phy level errors from the MIPI-CSI block on the CX3.

See Also

CyU3PMipicsiGetErrors

4.31.2 Field Documentation

4.31.2.1 uint8_t crcErrCnt

CRC Error Count

4.31.2.2 uint8_t ctlErrCnt

Control Error (Incorrect Line State Sequence) Count

4.31.2.3 uint8_t eidErrCnt

Unsupported Packet ID Error Count

4.31.2.4 uint8_t frmErrCnt

Framing Error Count

4.31.2.5 uint8_t mdlErrCnt

Multi-Data Lane Sync Byte Error Count

4.31.2.6 uint8_t recrErrCnt

Recoverable Packet Header Error Count

4.31.2.7 uint8_t recSyncErrCnt

Recoverable Sync Byte Error Count

4.31.2.8 uint8_t unrcErrCnt

Unrecoverable Packet Header Error Count

4.31.2.9 uint8_t unrSyncErrCnt

Unrecoverable Sync Byte Error Count

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

- firmware/u3p_firmware/inc/cyu3mipicsi.h

4.32 CyU3POtgConfig_t Struct Reference

OTG configuration information.

`#include <cyu3usbotg.h>`

Data Fields

- CyU3POtgMode_t otgMode
- CyU3POtgChargerDetectMode_t chargerMode
- CyU3POtgEventCallback_t cb

4.32.1 Detailed Description

OTG configuration information.

Description

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

See Also

CyU3POtgMode_t
CyU3POtgEventCallback_t
CyU3POtgStart

4.32.2 Field Documentation

4.32.2.1 CyU3POtgEventCallback_t cb

OTG event callback function.

4.32.2.2 CyU3POtgChargerDetectMode_t chargerMode

Charger detect mode.

4.32.2.3 CyU3POtgMode_t otgMode

USB port mode of operation.

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

- firmware/u3p_firmware/inc/cyu3usbotg.h

4.33 CyU3PPibClock_t Struct Reference

Clock configuration information for the PIB block.

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

Data Fields

- uint16_t clkDiv
- CyBool_t isHalfDiv
- CyBool_t isDllEnable
- CyU3PSysClockSrc_t clkSrc

4.33.1 Detailed Description

Clock configuration information for the PIB block.

Description

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

The default values used by the driver are:

clkDiv = 2, isHalfDiv = CyFalse, isDllEnable = CyFalse, and clkSrc = CY_U3P_SYS_CLK.

See Also

CyU3PSysClockSrc_t
CyU3PPibInit

4.33.2 Field Documentation

4.33.2.1 uint16_t clkDiv

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

4.33.2.2 CyU3PSysClockSrc_t clkSrc

The clock source to be used.

4.33.2.3 CyBool_t isDllEnable

This enables or disable the PIB DLL control.

4.33.2.4 CyBool_t isHalfDiv

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

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

- firmware/u3p_firmware/inc/cyu3pib.h

4.34 CyU3PSdioCardRegs Struct Reference

SDIO Card Generic Information and CCCR registers.

`#include <cyu3sib.h>`

Data Fields

- uint8_t isMemoryPresent
- uint8_t numberOfFunctions
- uint8_t CCCRVersion
- uint8_t sdioVersion
- uint8_t cardCapability
- uint8_t cardSpeed
- uint16_t fn0BlockSize
- uint32_t addrCIS
- uint16_t manufacturerId
- uint16_t manufacturerInfo
- uint8_t uhsSupport
- uint8_t supportsAsyncIntr

4.34.1 Detailed Description

SDIO Card Generic Information and CCCR registers.

Description

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

4.34.2 Field Documentation

4.34.2.1 uint32_t addrCIS

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

4.34.2.2 uint8_t cardCapability

Sdio Card Capability register from the CCCR

4.34.2.3 uint8_t cardSpeed

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

4.34.2.4 uint8_t CCCRVersion

CCCR Format Version Number as defined by SDIO spec

4.34.2.5 uint16_t fn0BlockSize

Function 0 Block Size

4.34.2.6 uint8_t isMemoryPresent

```
Is Memory is present on the SDIO. 1 in case of a Combo card,
```

0 for I/O only cards.

4.34.2.7 uint16_t manufacturerId

Manufacturer ID

4.34.2.8 uint16_t manufacturerInfo

Manufacturer information

4.34.2.9 uint8_t numberOfFunctions

Number of I/O Functions present on the card

4.34.2.10 uint8_t sdioVersion

```
Lower 4 bits define SDIO Version supported by the card.
```

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

4.34.2.11 uint8_t supportsAsyncIntr

```
Interrupt Extension byte from CCCR SDIO 3.0 Asynchronous Interrupt
```

support information.

4.34.2.12 uint8_t uhsSupport

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

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

- firmware/u3p_firmware/inc/cyu3sib.h

4.35 CyU3PSibDevInfo Struct Reference

Storage (SD/MMC) device properties.

`#include <cyu3sib.h>`

Data Fields

- CyU3PSibDevType cardType
- uint32_t clkRate
- uint32_t numBlks
- uint32_t eraseSize
- uint16_t blkLen
- uint16_t ccc
- uint8_t removable
- uint8_t writeable
- uint8_t locked
- uint8_t ddrMode
- uint8_t opVoltage
- uint8_t busWidth
- uint8_t numUnits

4.35.1 Detailed Description

Storage (SD/MMC) device properties.

Description

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

See Also

CyU3PSibDevType

4.35.2 Field Documentation

4.35.2.1 uint16_t blkLen

Current block size setting for the device.

4.35.2.2 uint8_t busWidth

Current bus width setting for the device.

4.35.2.3 CyU3PSibDevType cardType

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

4.35.2.4 uint16_t ccc

Card command classes (CCC) from the CSD register.

4.35.2.5 uint32_t clkRate

Current operating clock frequency for the device.

4.35.2.6 uint8_t ddrMode

Whether DDR clock mode is being used for this device.

4.35.2.7 uint32_t eraseSize

The erase unit size in bytes for this device.

4.35.2.8 uint8_t locked

Identifies whether the storage device is password locked.

4.35.2.9 uint32_t numBlks

Number of blocks of the storage device.

4.35.2.10 uint8_t numUnits

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

4.35.2.11 uint8_t opVoltage

Current operating voltage setting for the device.

4.35.2.12 uint8_t removable

Indicates if the storage device is a removable device.

4.35.2.13 uint8_t writeable

Whether the storage device is write enabled.

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

- firmware/u3p_firmware/inc/cyu3sib.h

4.36 CyU3PSibIntfParams Struct Reference

Storage interface control parameters.

`#include <cyu3sib.h>`

Data Fields

- uint8_t resetGpio
- CyBool_t rstActHigh
- uint8_t cardDetType
- uint8_t writeProtEnable
- uint8_t lowVoltage
- uint8_t voltageSwGpio
- CyBool_t lvGpioState
- uint8_t useDdr
- CyU3PSibOpFreq maxFreq
- uint8_t cardInitDelay

4.36.1 Detailed Description

Storage interface control parameters.

Description

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

4.36.2 Field Documentation

4.36.2.1 uint8_t cardDetType

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

4.36.2.2 uint8_t cardInitDelay

```
Initialization delay (in ms) to allow card to stabilize. Some
```

storage devices appear to need some time to start responding after initial power up.

4.36.2.3 uint8_t lowVoltage

Enable/disable low voltage operation on the port.

4.36.2.4 CyBool_t lvGpioState

```
State of the GPIO that sets the S-port to low voltage:
```

CyTrue : GPIO=0 => 3.3 V, GPIO=1 => 1.8 V. CyFalse: GPIO=1 => 3.3 V, GPIO=0 => 1.8 V.

4.36.2.5 CyU3PSibOpFreq maxFreq

Maximum operating frequency for thr S port.

4.36.2.6 uint8_t resetGpio

```
GPIO to be used for controlling power/reset to the storage device.
```

Set to 255 if no GPIO is to be used.

4.36.2.7 CyBool_t rstActHigh

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

4.36.2.8 uint8_t useDdr

Whether to enable DDR clock for the S port.

4.36.2.9 uint8_t voltageSwGpio

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

to be used.

4.36.2.10 uint8_t writeProtEnable

```
Indicates whether device specified write protection is allowed
```

for this S port.

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

- firmware/u3p_firmware/inc/cyu3sib.h

4.37 CyU3PSibLunInfo Struct Reference

Logical unit properties.

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

Data Fields

- uint32_t startAddr
- uint32_t numBlocks
- uint32_t blockSize
- uint8_t valid
- uint8_t writeable
- CyU3PSibLunLocation location
- CyU3PSibLunType type

4.37.1 Detailed Description

Logical unit properties.

Description

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

See Also

CyU3PSibLunType
CyU3PSibLunLocation

4.37.2 Field Documentation

4.37.2.1 uint32_t blockSize

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

the device block size.

4.37.2.2 CyU3PSibLunLocation location

Location of the logical unit.

4.37.2.3 uint32_t numBlocks

Size of the partition in blocks.

4.37.2.4 uint32_t startAddr

Starting address for the logical unit within the device.

4.37.2.5 CyU3PSibLunType type

Identifies the type of the logical unit.

4.37.2.6 uint8_t valid

Whether this partition exists on the storage device.

4.37.2.7 uint8_t writeable

```
Whether the unit is write enabled. The application can define separate
```

write permissions for each unit.

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

- firmware/u3p_firmware/inc/cyu3sib.h

4.38 CyU3PSpiConfig_t Struct Reference

Structure defining the configuration of SPI interface.

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

Data Fields

- CyBool_t isLsbFirst
- CyBool_t cpol
- CyBool_t cpha
- CyBool_t ssnPol
- CyU3PSpiSsnCtrl_t ssnCtrl
- CyU3PSpiSsnLagLead_t leadTime
- CyU3PSpiSsnLagLead_t lagTime
- uint32_t clock
- uint8_t wordLen

4.38.1 Detailed Description

Structure defining the configuration of SPI interface.

Description

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

See Also

CyU3PSpiSsnCtrl_t
CyU3PSpiSclkParam_t
CyU3PSpiSetConfig

4.38.2 Field Documentation

4.38.2.1 uint32_t clock

SPI clock frequency in Hz.

4.38.2.2 CyBool_t cpha

```
Clock phase - CyFalse(0): Slave samples at idle-active edge,
```

CyTrue(1): Slave samples at active-idle edge

4.38.2.3 CyBool_t cpol

Clock polarity - CyFalse(0): SCK idles low, CyTrue(1): SCK idles high

4.38.2.4 CyBool_t isLsbFirst

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

4.38.2.5 CyU3PSpiSsnLagLead_t lagTime

```
Time between the last SCK edge to SSN de-assertion. This is at the
```

end of a transfer and is valid only when the hardware controls the SSN. When CPHA = 1, lag time cannot be zero.

4.38.2.6 CyU3PSpiSsnLagLead_t leadTime

```
Time between SSN assertion and first SCLK edge. This is at the
```

beginning of a transfer and is valid only when the hardware controls the SSN. A lead time of zero is not supported.

4.38.2.7 CyU3PSpiSsnCtrl_t ssnCtrl

SSN control method.

4.38.2.8 CyBool_t ssnPol

```
Polarity of SSN line - CyFalse (0): SSN is active low,
```

CyTrue (1): SSN is active high.

4.38.2.9 uint8_t wordLen

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

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

- firmware/u3p_firmware/inc/cyu3spi.h

4.39 CyU3PSysClockConfig_t Struct Reference

Clock configuration for FX3 CPU, DMA and register access.

`#include <cyu3system.h>`

Data Fields

- CyBool_t setSysClk400
- uint8_t cpuClkDiv
- uint8_t dmaClkDiv
- uint8_t mmioClkDiv
- CyBool_t useStandbyClk
- CyU3PSysClockSrc_t clkSrc

4.39.1 Detailed Description

Clock configuration for FX3 CPU, DMA and register access.

Description

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

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

The useStandbyClk parameter specifies whether a 32KHz clock has been supplied on the CLKIN_32 pin of the device. This clock is the standby clock for the device. If this pin is not connected, then the device internally generates a clock from the SYS_CLK.

The setSysClk400 parameter specifies whether the FX3 device's master clock is to be set to a frequency greater than 400 MHz. By default, the FX3 master clock is set to 384 MHz when using a 19.2 MHz crystal or clock source.

This frequency setting may lead to DMA overflow errors on the GPIF, if the GPIF is configured as 32-bit wide and is running at 100 MHz. Setting this parameter will switch the master clock frequency to 403.2 MHz during the CyU3PDeviceInit call.

This structure is passed as parameter to CyU3PDeviceInit call.

See Also

CyU3PSysClockSrc_t
CyU3PDeviceInit
CyU3PDeviceGetSysClkFreq

4.39.2 Field Documentation

4.39.2.1 CyU3PSysClockSrc_t clkSrc

Clock source for CPU clocking.

4.39.2.2 uint8_t cpuClkDiv

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

4.39.2.3 uint8_t dmaClkDiv

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

4.39.2.4 uint8_t mmioClkDiv

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

4.39.2.5 CyBool_t setSysClk400

```
Whether the FX3 master (System) clock is to be set to a frequency
```

greater than 400 MHz. This is required to be set to True if the GPIF is running in 32-bit mode at 100 MHz.

4.39.2.6 CyBool_t useStandbyClk

Whether the 32 KHz standby clock is supplied.

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

- firmware/u3p_firmware/inc/cyu3system.h

4.40 CyU3PUartConfig_t Struct Reference

Configuration parameters for the UART interface.

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

Data Fields

- CyBool_t txEnable
- CyBool_t rxEnable
- CyBool_t flowCtrl
- CyBool_t isDma
- CyU3PUartBaudrate_t baudRate
- CyU3PUartStopBit_t stopBit
- CyU3PUartParity_t parity

4.40.1 Detailed Description

Configuration parameters for the UART interface.

Description

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

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

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

See Also

CyU3PUartBaudrate_t
CyU3PUartStopBit_t
CyU3PUartParity_t
CyU3PUartSetConfig

4.40.2 Field Documentation

4.40.2.1 CyU3PUartBaudrate_t baudRate

Baud rate for data transfer.

4.40.2.2 CyBool_t flowCtrl

Enable hardware flow control for both RX and TX.

4.40.2.3 CyBool_t isDma

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

4.40.2.4 CyU3PUartParity_t parity

Parity configuration.

4.40.2.5 CyBool_t rxEnable

Enable the receiver.

4.40.2.6 CyU3PUartStopBit_t stopBit

The number of stop bits appended.

4.40.2.7 CyBool_t txEnable

Enable the transmitter.

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

- firmware/u3p_firmware/inc/cyu3uart.h

4.41 CyU3PUsbDescrPtrs Struct Reference

Pointer to the various descriptors.

`#include <cyfx3usb.h>`

Data Fields

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

4.41.1 Detailed Description

Pointer to the various descriptors.

Description

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

4.41.2 Field Documentation

4.41.2.1 uint8_t* usbConfigDesc_p

Pointer to config desc of device

4.41.2.2 uint8_t* usbDevDesc_p

Pointer to device desc of device

4.41.2.3 uint8_t* usbDevQualDesc_p

Pointer to device qualifier desc of device

4.41.2.4 uint8_t* usbFSConfigDesc_p

Pointer to FULL SPEED speed configuration desc of device

4.41.2.5 uint8_t* usbHSConfigDesc_p

Pointer to HIGH SPEED speed configuration desc of device

4.41.2.6 uint8_t* usbOtherSpeedConfigDesc_p

Pointer to other speed configuration desc of device

4.41.2.7 uint8_t* usbSSBOSDesc_p

Pointer to Super speed BOS descriptor

4.41.2.8 uint8_t* usbSSConfigDesc_p

Pointer to SUPER SPEED speed configuration desc of device

4.41.2.9 uint8_t* usbSSDevDesc_p

Pointer to SS device desc of device

4.41.2.10 uint8_t* usbStringDesc_p[CY_FX3_USB_MAX_STRING_DESC_INDEX]

Array of pointers to string descriptors.

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

- firmware/boot_fw/include/cyfx3usb.h

4.42 CyU3PUsbHostConfig_t Struct Reference

USB host mode configuration information.

`#include <cyu3usbhost.h>`

Data Fields

- CyBool_t ep0LowLevelControl
- CyU3PUsbHostEventCb_t eventCb
- CyU3PUsbHostXferCb_t xferCb

4.42.1 Detailed Description

USB host mode configuration information.

Description

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

See Also

CyU3PUsbHostStart

4.42.2 Field Documentation

4.42.2.1 CyBool_t ep0LowLevelControl

```
Whether to enable EP0 low level control. If CyFalse, the EP0
```

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

4.42.2.2 CyU3PUsbHostEventCb_t eventCb

Event callback function for USB host stack.

4.42.2.3 CyU3PUsbHostXferCb_t xferCb

EP transfer completion callback for USB host stack.

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

- firmware/u3p_firmware/inc/cyu3usbhost.h

4.43 CyU3PUsbHostEpConfig_t Struct Reference

Host mode endpoint configuration structure.

`#include <cyu3usbhost.h>`

Data Fields

- CyU3PUsbEpType_t type
- uint8_t mult
- uint16_t maxPktSize
- uint8_t pollingRate
- uint16_t fullPktSize
- CyBool_t isStreamMode

4.43.1 Detailed Description

Host mode endpoint configuration structure.

Description

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

See Also

CyU3PUsbHostEpAdd

4.43.2 Field Documentation

4.43.2.1 uint16_t fullPktSize

```
This is used for DMA packet boundary identification. If the DMA
```

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

4.43.2.2 CyBool_t isStreamMode

```
Enable stream mode. This means that the EP is always active. This setting
```

is valid only for an IN EP, and should be CyFalse for EP0 and OUT endpoints. When the flag is set, an IN EP will send IN tokens and collect data whenever there is a free buffer on the DMA channel.

4.43.2.3 uint16_t maxPktSize

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

USB specification.

4.43.2.4 uint8_t mult

```
Number of packets to be transferred per micro-frame. This should be 1
```

for bulk, control and interrupt endpoints; and 1, 2, or 3 for isochronous endpoints.

4.43.2.5 uint8_t pollingRate

```
Rate at which the endpoint has to be polled in ms. Zero will indicate
```

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

4.43.2.6 CyU3PUsbEpType_t type

Type of endpoint to be created.

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

- firmware/u3p_firmware/inc/cyu3usbhost.h

Chapter 5

File Documentation

5.1 firmware/boot_fw/include/cyfx3device.h File Reference

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

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

Data Structures

- struct CyFx3BootIoMatrixConfig_t

  Defines the IO matrix configuration parameters.

Macros

- #define CY_FX3_BOOT_ITCM_BASE (0x00000000)

  Base address of the I-TCM region.
- #define CY_FX3_BOOT_ITCM_END (CY_FX3_BOOT_ITCM_BASE + 0x4000)

  End address of the I-TCM region.
- #define CY_FX3_BOOT_SYSMEM_BASE (0x40000000)

  Base address of the SYSMEM region.
- #define CY_FX3_BOOT_SYSMEM_END (CY_FX3_BOOT_SYSMEM_BASE + 0x80000)

  End address of the SYSMEM region. This value is valid for the CYUSB3014 part. Some other FX3 parts could have smaller SYSMEM regions.
- #define CY_FX3_BOOT_SYSMEM_BASE1 (CY_FX3_BOOT_SYSMEM_BASE + 0x2000)

  Start address of the usable range of SYSMEM.
- #define CY_FX3_BOOT_NO_WAIT (0x00)

  Option that specifies that a polling API should return immediately after checking the status of the transfer.
- #define CY_FX3_BOOT_WAIT_FOREVER (0xFFFFFFFF)

  Option that specifies that a polling API should wait as long as required for a transfer to complete.

Typedefs

- typedef enum CyFx3BootSysClockSrc_t CyFx3BootSysClockSrc_t

  Clock source for a peripheral block.
- typedef CyU3PPartNumber_t CyFx3PartNumber_t

  Enumeration of EZ-USB FX3 part numbers.
- typedef struct CyFx3BootIoMatrixConfig_t CyFx3BootIoMatrixConfig_t

  Defines the IO matrix configuration parameters.

Enumerations

- enum CyFx3BootSysClockSrc_t { CY_FX3_BOOT_SYS_CLK_BY_16 = 0, CY_FX3_BOOT_SYS_CLK_BY_4, CY_FX3_BOOT_SYS_CLK_BY_2, CY_FX3_BOOT_SYS_CLK, CY_FX3_BOOT_NUM_CLK_SRC }

  Clock source for a peripheral block.

Functions

- void CyFx3BootDeviceInit (CyBool_t setFastSysClk)

  This function initializes the FX3 device.
- void CyFx3BootJumpToProgramEntry (uint32_t address)

  Transfer control to the specified address.
- CyFx3BootErrorCode_t CyFx3BootDeviceConfigureIOMatrix (CyFx3BootIoMatrixConfig_t *cfg_p)

  Configure the IO matrix for the device.
- void CyFx3BootWatchdogConfigure (CyBool_t enable, uint32_t period)

  Enable/disable the watchdog timer.
- void CyFx3BootWatchdogClear (void)

  Clear the watchdog timer to prevent device reset.
- CyFx3PartNumber_t CyFx3BootGetPartNumber (void)

  Get the part number of the FX3 device in use.
- void CyFx3BootRetainGpioState (void)

  Request to keep the GPIO block powered ON across control transfer to the full firmware.
- CyFx3BootErrorCode_t CyFx3BootGpioOverride (uint8_t pinNumber)

  Override a specific FX3 pin as a GPIO.
- CyFx3BootErrorCode_t CyFx3BootGpioRestore (uint8_t pinNumber)

  Restore the standard function of a pin.

5.1.1 Detailed Description

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

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

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

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

1. USB device mode
2. SPI
3. I2C
4. UART
5. GPIO

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

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

5.1.2 FX3 Memory Regions

ITCM and SYSMEM memory address ranges on the FX3 device.

Description

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

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

5.1.3 Typedef Documentation

5.1.3.1 typedef struct CyFx3BootIoMatrixConfig_t CyFx3BootIoMatrixConfig_t

Defines the IO matrix configuration parameters.

Description

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

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

See Also

CyFx3BootDeviceConfigureIOMatrix
CyFx3BootGpioOverride

5.1.3.2 typedef enum CyFx3BootSysClockSrc_t CyFx3BootSysClockSrc_t

Clock source for a peripheral block.

Description

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

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

5.1.3.3 typedef CyU3PPartNumber_t CyFx3PartNumber_t

Enumeration of EZ-USB FX3 part numbers.

Description

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

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

See Also

CyFx3BootGetPartNumber

5.1.4 Enumeration Type Documentation

5.1.4.1 enum CyFx3BootSysClockSrc_t

Clock source for a peripheral block.

Description

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

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

Enumerator

CY_FX3_BOOT_SYS_CLK_BY_16 SYS_CLK divided by 16.

CY_FX3_BOOT_SYS_CLK_BY_4 SYS_CLK divided by 4.

CY_FX3_BOOT_SYS_CLK_BY_2 SYS_CLK divided by 2.

CY_FX3_BOOT_SYS_CLK SYS_CLK itself.

CY_FX3_BOOT_NUM_CLK_SRC Number of clock source enumerations.

5.1.5 Function Documentation

5.1.5.1 CyFx3BootErrorCode_t CyFx3BootDeviceConfigureIOMatrix (CyFx3BootIoMatrixConfig_t * *cfg_p*)

Configure the IO matrix for the device.

Description

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

This API completes the IO configuration based on the user specified parameters. Any pin that is not used as part of the GPIF or serial peripheral interfaces can be used as a GPIO.

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

Return value

CY_FX3_BOOT_SUCCESS - When the IO configuration is successful.

CY_FX3_BOOT_ERROR_NOT_SUPPORTED - the FX3 part in use does not support the desired configuration.

CY_FX3_BOOT_ERROR_BAD_ARGUMENT - If some configuration value is invalid.

See Also

CyFx3BootIoMatrixConfig_t

Parameters

cfg_p	Pointer to configuration structure.

5.1.5.2 void CyFx3BootDeviceInit (CyBool_t *setFastSysClk*)

This function initializes the FX3 device.

Description

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

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

Parameters

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.

5.1.5.3 CyFx3PartNumber_t CyFx3BootGetPartNumber (void)

Get the part number of the FX3 device in use.

Description

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

Return value

Part number of the FX3 device in use.

See Also

CyFx3PartNumber_t

5.1.5.4 CyFx3BootErrorCode_t CyFx3BootGpioOverride (uint8_t *pinNumber*)

Override a specific FX3 pin as a GPIO.

Description

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

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

Return value

CY_FX3_BOOT_SUCCESS if the pin override is successful.

CY_FX3_BOOT_ERROR_BAD_ARGUMENT if the pin specified is not valid.

See Also

CyFx3BootDeviceConfigureIOMatrix
CyFx3BootGpioRestore

Parameters

pinNumber	Pin number to be over-ridden as a simple GPIO.

5.1.5.5 CyFx3BootErrorCode_t CyFx3BootGpioRestore (uint8_t *pinNumber*)

Restore the standard function of a pin.

Description

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

Return value

CY_FX3_BOOT_SUCCESS if the pin restore is successful.

CY_FX3_BOOT_ERROR_BAD_ARGUMENT if the pin specified is not valid.

See Also

CyFx3BootGpioOverride

Parameters

pinNumber	Pin number to be restored to default function.

5.1.5.6 void CyFx3BootJumpToProgramEntry (uint32_t *address*)

Transfer control to the specified address.

Description

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

Parameters

address	The program entry address

5.1.5.7 void CyFx3BootRetainGpioState (void)

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

Description

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

Return value

None

5.1.5.8 void CyFx3BootWatchdogClear (void)

Clear the watchdog timer to prevent device reset.

Description

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

Return value

None

See Also

CyFx3BootWatchdogConfigure

5.1.5.9 void CyFx3BootWatchdogConfigure (CyBool_t *enable*, uint32_t *period*)

Enable/disable the watchdog timer.

Description

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

Return value

None

See Also

CyFx3BootWatchdogClear

Parameters

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

5.2 firmware/boot_fw/include/cyfx3error.h File Reference

This file lists the various error codes that can be returned by the APIs in the FX3 boot library.

```
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

Typedefs

- typedef enum CyFx3BootErrorCode_t CyFx3BootErrorCode_t

  List of error codes returned by the boot APIs.

Enumerations

- enum CyFx3BootErrorCode_t {
  CY_FX3_BOOT_SUCCESS = 0x0, CY_FX3_BOOT_ERROR_BAD_ARGUMENT, CY_FX3_BOOT_ERROR_NULL_POINTER, CY_FX3_BOOT_ERROR_TIMEOUT,
  CY_FX3_BOOT_ERROR_NOT_SUPPORTED, CY_FX3_BOOT_ERROR_NOT_CONFIGURED, CY_FX3_BOOT_ERROR_BAD_DESCRIPTOR_TYPE, CY_FX3_BOOT_ERROR_XFER_FAILURE,
  CY_FX3_BOOT_ERROR_NO_REENUM_REQUIRED, CY_FX3_BOOT_ERROR_NOT_STARTED, CY_FX3_BOOT_ERROR_MEMORY_ERROR, CY_FX3_BOOT_ERROR_ABORTED,
  CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR, CY_FX3_BOOT_ERROR_FAILURE }

List of error codes returned by the boot APIs.

5.2.1 Detailed Description

This file lists the various error codes that can be returned by the APIs in the FX3 boot library.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum CyFx3BootErrorCode_t

List of error codes returned by the boot APIs.

Enumerator

CY_FX3_BOOT_SUCCESS Success.

CY_FX3_BOOT_ERROR_BAD_ARGUMENT One or more parameters to a function are invalid.

CY_FX3_BOOT_ERROR_NULL_POINTER A null pointer has been passed in unexpectedly.

CY_FX3_BOOT_ERROR_TIMEOUT Timeout on relevant operation.

CY_FX3_BOOT_ERROR_NOT_SUPPORTED Operation requested is not supported in current mode.

CY_FX3_BOOT_ERROR_NOT_CONFIGURED The peripheral block being accessed has not been configured.

CY_FX3_BOOT_ERROR_BAD_DESCRIPTOR_TYPE Invalid USB descriptor type.

CY_FX3_BOOT_ERROR_XFER_FAILURE Data Transfer failed.

CY_FX3_BOOT_ERROR_NO_REENUM_REQUIRED Indicates that the booter has successfully configured the FX3 device after control was transferred from the full firmware application. The user need not go through the cycle of setting the descriptors and issuing connect call if this error code is returned by the CyFx3BootUsbStart () API.

CY_FX3_BOOT_ERROR_NOT_STARTED Indicates that the block being configured has not been initialized.

CY_FX3_BOOT_ERROR_MEMORY_ERROR Indicates that the API was unable to find enough memory to copy the descriptor set through the CyFx3BootUsbSetDesc() API.

CY_FX3_BOOT_ERROR_ABORTED Indicates that the USB control request being processed has been aborted due to a USB reset or a new control request.

CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR Indicates that the address passed into a DMA transfer API is invalid (could be a TCM address).

CY_FX3_BOOT_ERROR_FAILURE Generic error code indicating any other failure.

5.3 firmware/boot_fw/include/cyfx3gpio.h File Reference

The file defines the data structures and APIs that are provided for making use of FX3 GPIOs.

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

Data Structures

- struct CyFx3BootGpioSimpleConfig_t

  Configuration information for simple GPIOs.

Typedefs

- typedef enum CyFx3BootGpioIntrMode_t CyFx3BootGpioIntrMode_t

  List of GPIO interrupt modes.
- typedef struct CyFx3BootGpioSimpleConfig_t CyFx3BootGpioSimpleConfig_t

  Configuration information for simple GPIOs.

Enumerations

- enum CyFx3BootGpioIntrMode_t { CY_FX3_BOOT_GPIO_NO_INTR = 0, CY_FX3_BOOT_GPIO_INTR_POS_EDGE, CY_FX3_BOOT_GPIO_INTR_NEG_EDGE, CY_FX3_BOOT_GPIO_INTR_BOTH_EDGE, CY_FX3_BOOT_GPIO_INTR_LOW_LEVEL, CY_FX3_BOOT_GPIO_INTR_HIGH_LEVEL }

  List of GPIO interrupt modes.

Functions

- void CyFx3BootGpioInit (void)

  Initializes the GPIO block.
- void CyFx3BootGpioDeInit (void)

  De-initializes the GPIO block.
- CyFx3BootErrorCode_t CyFx3BootGpioSetSimpleConfig (uint8_t gpioId, CyFx3BootGpioSimpleConfig_t *cfg_p)

  Configures a simple GPIO.
- void CyFx3BootGpioDisable (uint8_t gpioId)

  Disables a GPIO pin.
- CyFx3BootErrorCode_t CyFx3BootGpioGetValue (uint8_t gpioId, CyBool_t *value_p)

  Query the state of GPIO input.
- CyFx3BootErrorCode_t CyFx3BootGpioSetValue (uint8_t gpioId, CyBool_t value)

  Set the state of GPIO pin output.

5.3.1 Detailed Description

The file defines the data structures and APIs that are provided for making use of FX3 GPIOs. **Description**

Any unused pin on the FX3 device can be used as a GPIO. The Boot API library only allows for these pins to be used as standard input or output pins.

5.3.2 Typedef Documentation

5.3.2.1 typedef enum CyFx3BootGpioIntrMode_t CyFx3BootGpioIntrMode_t

List of GPIO interrupt modes.

Description

FX3 GPIOs can be configured to trigger interrupts when a desired transition or signal level is detected on the input signal.

Note

Interrupts are not supported in this release of booter. The interrupt mode for all GPIOs need to be set to CY_FX3_BOOT_GPIO_NO_INTR.

5.3.2.2 typedef struct CyFx3BootGpioSimpleConfig_t CyFx3BootGpioSimpleConfig_t

Configuration information for simple GPIOs.

Description

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

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

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

See Also

CyFx3BootGpioIntrMode_t

5.3.3 Enumeration Type Documentation

5.3.3.1 enum CyFx3BootGpioIntrMode_t

List of GPIO interrupt modes.

Description

FX3 GPIOs can be configured to trigger interrupts when a desired transition or signal level is detected on the input signal.

Note

Interrupts are not supported in this release of booter. The interrupt mode for all GPIOs need to be set to CY_FX3_BOOT_GPIO_NO_INTR.

Enumerator

CY_FX3_BOOT_GPIO_NO_INTR No Interrupt is triggered.

CY_FX3_BOOT_GPIO_INTR_POS_EDGE Interrupt is triggered on positive edge of input.

CY_FX3_BOOT_GPIO_INTR_NEG_EDGE Interrupt is triggered on negative edge of input.

CY_FX3_BOOT_GPIO_INTR_BOTH_EDGE Interrupt is triggered on both edges of input.

CY_FX3_BOOT_GPIO_INTR_LOW_LEVEL Interrupt is triggered when input is low.

CY_FX3_BOOT_GPIO_INTR_HIGH_LEVEL Interrupt is triggered when input is high.

5.3.4 Function Documentation

5.3.4.1 void CyFx3BootGpioDeInit (void)

De-initializes the GPIO block.

Description

This function powers off the GPIO block on the FX3 device.

Return value

None

See Also

CyFx3BootGpioInit

5.3.4.2 void CyFx3BootGpioDisable (uint8_t *gpioId*)

Disables a GPIO pin.

Description

This function is used to disable the use of a pin as a GPIO. Both input and output stages on the pin will be disabled after calling this API.

Return value

None

See Also

CyFx3BootGpioSetSimpleConfig

Parameters

gpioId	GPIO ID to be disabled

5.3.4.3 CyFx3BootErrorCode_t CyFx3BootGpioGetValue (uint8_t *gpioId*, CyBool_t * *value_p*)

Query the state of GPIO input.

Description

Get the current signal level on a GPIO input pin. This API can also be used to retrieve the current value being driven out on a output pin.

Return value

CY_FX3_BOOT_SUCCESS - if the operation is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - if value_p is NULL.

CY_FX3_BOOT_ERROR_NOT_CONFIGURED - if the pin is not selected as a GPIO in the IOMATRIX.

See Also

CyFx3BootGpioSetSimpleConfig
CyFx3BootGpioSetValue

Parameters

gpioId	GPIO id to be queried.
value_p	Output parameter that will be filled with the GPIO value.

5.3.4.4 void CyFx3BootGpioInit (void)

Initializes the GPIO block.

Description

This function should be called before any other GPIO APIs can be called. This powers up the GPIO block and sets up the API data structures.

Return value

None

See Also

CyFx3BootGpioDeInit

5.3.4.5 CyFx3BootErrorCode_t CyFx3BootGpioSetSimpleConfig (uint8_t *gpioId*, CyFx3BootGpioSimpleConfig_t * *cfg_p*)

Configures a simple GPIO.

Description

This function is used to configure and enable a simple GPIO pin. This function needs to be called before using the simple GPIO.

Return value

CY_FX3_BOOT_SUCCESS - if the operation is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - if cfg_p is NULL.

CY_FX3_BOOT_ERROR_NOT_CONFIGURED - if the pin has not been selected as a GPIO in the IOMATRIX.

See Also

CyFx3BootGpioSimpleConfig_t
CyFx3BootDeviceConfigureIOMatrix
CyFx3BootGpioDisable

Parameters

gpioId	GPIO id to be configured.
cfg_p	Configuration parameters for the GPIO.

5.3.4.6 CyFx3BootErrorCode_t CyFx3BootGpioSetValue (uint8_t *gpioId*, CyBool_t *value*)

Set the state of GPIO pin output.

Description

This function updates the output state of a GPIO pin, and is valid only for GPIOs configured as output. The output value is updated by this function, but the driveLowEn and driveHighEn configuration parameters will determine whether the pin is driven actively or tri-stated.

Return value

CY_FX3_BOOT_SUCCESS - if the operation is successful.

CY_FX3_BOOT_ERROR_NOT_CONFIGURED - if the pin is not selected as a GPIO in the IOMATRIX.

See Also

CyFx3BootGpioSetSimpleConfig
CyFx3BootGpioGetValue

Parameters

gpioId	GPIO id to be modified.
value	Value to set on the GPIO pin.

5.4 firmware/boot_fw/include/cyfx3i2c.h File Reference

The I2C interface driver in the FX3 Boot Firmware provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

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

Data Structures

- struct CyFx3BootI2cConfig_t

  Structure defining the configuration of the I2C interface.
- struct CyFx3BootI2cPreamble_t

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

Typedefs

- typedef struct CyFx3BootI2cConfig_t CyFx3BootI2cConfig_t

  Structure defining the configuration of the I2C interface.
- typedef struct CyFx3BootI2cPreamble_t CyFx3BootI2cPreamble_t

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

Functions

- CyFx3BootErrorCode_t CyFx3BootI2cInit (void)

  Starts the I2C interface block.
- void CyFx3BootI2cDeInit (void)

  Stops the I2C module.
- CyFx3BootErrorCode_t CyFx3BootI2cSetConfig (CyFx3BootI2cConfig_t *config)

  Sets the I2C interface parameters.
- CyFx3BootErrorCode_t CyFx3BootI2cSendCommand (CyFx3BootI2cPreamble_t *preamble, uint32_t byteCount, CyBool_t isRead)

  Perform a read or write operation to the I2C slave.
- CyFx3BootErrorCode_t CyFx3BootI2cTransmitBytes (CyFx3BootI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount)

  Writes a small number of bytes to an I2C slave.
- CyFx3BootErrorCode_t CyFx3BootI2cReceiveBytes (CyFx3BootI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount)

  Reads a small number of bytes from an I2C slave.
- CyFx3BootErrorCode_t CyFx3BootI2cWaitForAck (CyFx3BootI2cPreamble_t *preamble, uint32_t retryCount)

  Poll an I2C slave until all of the preamble is ACKed.
- CyFx3BootErrorCode_t CyFx3BootI2cDmaXferData (CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout)

  This function is used to setup a dma from CPU to I2C or vice versa.

5.4.1 Detailed Description

The I2C interface driver in the FX3 Boot Firmware provides a set of APIs that allow the configuration of the I2C interface properties and data exchange with one or more I2C slave devices.

5.4.2 Typedef Documentation

5.4.2.1 typedef struct CyFx3BootI2cConfig_t CyFx3BootI2cConfig_t

Structure defining the configuration of the I2C interface.

Description

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

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

See Also

CyFx3BootI2cSetConfig

5.4.2.2 typedef struct CyFx3BootI2cPreamble_t CyFx3BootI2cPreamble_t

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

Description

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

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

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

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

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

Byte 3:

Bit 7 - 1: Slave address.

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

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

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

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

See Also

CyFx3BootI2cSetConfig

5.4.3 Function Documentation

5.4.3.1 void CyFx3BootI2cDeInit (void)

Stops the I2C module.

Description

This function disables and powers off the I2C interface. This function can be used to shut off the interface to save power when it is not in use.

Return value

None

See Also

CyFx3BootI2cInit

5.4.3.2 CyFx3BootErrorCode_t CyFx3BootI2cDmaXferData (CyBool_t *isRead*, uint32_t *address*, uint32_t *length*, uint32_t *timeout*)

This function is used to setup a dma from CPU to I2C or vice versa.

Description

This function is used to read/write data from/to an I2C slave when the I2C interface is configured in DMA mode. The CyFx3BootI2cSendCommand API should be used to address the slave and initiate the data transfer on the slave side.

This function is a blocking call which waits until the desired transfer is complete or the specified timeout duration has elapsed.

It is expected that the length of data being transferred is a multiple of 16 bytes.

Return value

CY_FX3_BOOT_SUCCESS - if the data transfer is successful.

CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR - if the address specified is not in the SYSMEM area.

CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error.

CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out.

See Also

CyFx3BootI2cSetConfig
CyFx3BootI2cSendCommand

Parameters

isRead	isRead=CyTrue for read transfers, and isRead=CyFalse for write transfers.
address	Address of the buffer from/to which data is to be transferred
length	Length of the data to be transferred. Should be a multiple of 16 bytes and is limited by the amount of data that the slave can transfer without delays.
timeout	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to wait until the transfer is complete.

5.4.3.3 CyFx3BootErrorCode_t CyFx3BootI2cInit (void)

Starts the I2C interface block.

Description

This function powers up the I2C interface block on the FX3 device and is expected to be the first I2C API function that is called by the application. IO configuration function is expected to have been called prior to this call.

This function also sets up the I2C interface at a default bit rate of 100KHz.

Return value

CY_FX3_BOOT_SUCCESS - If the block is initialized successfully.

See Also

CyFx3BootI2cSetConfig
CyFx3BootI2cDeinit

5.4.3.4 CyFx3BootErrorCode_t CyFx3BootI2cReceiveBytes (CyFx3BootI2cPreamble_t * *preamble*, uint8_t * *data*, uint32_t *byteCount*, uint32_t *retryCount*)

Reads a small number of bytes from an I2C slave.

Description

This function reads a few bytes one at a time from the I2C slave selected through the preamble parameter. The I2C interface should be configured in register (non-DMA) mode to make use of this function.

Return value

CY_FX3_BOOT_SUCCESS - if the TransmitBytes is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - if the preamble or data is NULL.

CY_FX3_BOOT_ERROR_XFER_FAILURE - if the transfer fails due to a bus error.

CY_FX3_BOOT_ERROR_TIMEOUT - if the transfer times out.

See Also

CyFx3BootI2cSetConfig
CyFx3BootI2cTransmitBytes

Parameters

preamble	Preamble to be sent out before the data transfer.
data	Pointer to buffer where the data is to be placed.
byteCount	Size of the transfer in bytes.
retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

5.4.3.5 CyFx3BootErrorCode_t CyFx3BootI2cSendCommand (CyFx3BootI2cPreamble_t * *preamble*, uint32_t *byteCount*, CyBool_t *isRead*)

Perform a read or write operation to the I2C slave.

Description

This function is used to send the extended preamble over the I2C bus, and is used to initiate a transfer when the I2C block is configured for DMA mode of transfer.

This function is used internally by the CyFx3BootI2cReceiveBytes and CyFx3BootI2cTransmitBytes APIs, and should not be called directly when the register mode of transfer is selected.

Return value

CY_FX3_BOOT_SUCCESS - When the SendCommand is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - When the preamble is NULL.

CY_FX3_BOOT_ERROR_TIMEOUT - When the transfer times out.

See Also

CyFx3BootI2cReceiveBytes
CyFx3BootI2cTransmitBytes

Parameters

preamble	Preamble information to be sent out.
byteCount	Size of the transfer in bytes.
isRead	Direction of transfer; CyTrue: Read, CyFalse: Write.

5.4.3.6 CyFx3BootErrorCode_t CyFx3BootI2cSetConfig (CyFx3BootI2cConfig_t * *config*)

Sets the I2C interface parameters.

Description

This function is used to configure the I2C master interface based on the desired baud rate and address length settings to talk to the desired slave. This function should be called repeatedly to change the settings if different settings are to be used to communicate with different slave devices.

Return value

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 - If the driver timed out while trying to clear the RX and TX FIFOs.

See Also

CyFx3BootI2cConfig_t

Parameters

config	I2C configuration settings.

5.4.3.7 CyFx3BootErrorCode_t CyFx3BootI2cTransmitBytes (CyFx3BootI2cPreamble_t * *preamble,* uint8_t * *data,* uint32_t *byteCount,* uint32_t *retryCount*)

Writes a small number of bytes to an I2C slave.

Description

This function is used to write data one byte at a time to an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode.

If a non-zero retryCount is specified, the API will retry a failing transfer until it succeeds or the specified count has been reached.

Return value

CY_FX3_BOOT_SUCCESS - if the TransmitBytes is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - if the preamble or data is NULL.

CY_FX3_BOOT_ERROR_XFER_FAILURE - if the transfer fails due to a bus error.

CY_FX3_BOOT_ERROR_TIMEOUT - if the transfer times out.

See Also

CyFx3BootI2cSetConfig
CyFx3BootI2cReceiveBytes

Parameters

preamble	Preamble to be sent out before the data transfer.
data	Pointer to buffer containing data to be written.
byteCount	Size of the transfer in bytes.
retryCount	Number of times to retry request if a byte is NAKed by the slave.

5.4.3.8 CyFx3BootErrorCode_t CyFx3BootI2cWaitForAck (CyFx3BootI2cPreamble_t * *preamble,* uint32_t *retryCount*)

Poll an I2C slave until all of the preamble is ACKed.

Description

This function waits for a ACK handshake from the slave, and can be used to ensure that the slave device has reached a desired state before issuing the next transaction or shutting the interface down. This function call returns when the specified handshake has been received or when the wait has timed out. The function call can be repeated on CY_FX3_BOOT_ERROR_TIMEOUT without any error recovery.

Return value

CY_FX3_BOOT_SUCCESS - if the device ACKs the preamble.

CY_FX3_BOOT_ERROR_NULL_POINTER - if the preamble is NULL.

CY_FX3_BOOT_ERROR_TIMEOUT - if a timeout occurs.

Parameters

preamble	Preamble to be sent out for polling the slave.
retryCount	Number of times to retry request if preamble is NAKed or an error is encountered.

5.5 firmware/boot_fw/include/cyfx3spi.h File Reference

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. This file defines the data structures and APIs that enable SPI slave access from the FX3 boot API library.

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

Data Structures

- struct CyFx3BootSpiConfig_t

  Structure defining the configuration of SPI interface.

Typedefs

- typedef enum CyFx3BootSpiSsnLagLead_t CyFx3BootSpiSsnLagLead_t

  Enumeration defining SSN lead and lag times with respect to SCK.
- typedef enum CyFx3BootSpiSsnCtrl_t CyFx3BootSpiSsnCtrl_t

  List of ways in which the SPI Slave Select (SSN) line can be controlled.
- typedef struct CyFx3BootSpiConfig_t CyFx3BootSpiConfig_t

  Structure defining the configuration of SPI interface.

Enumerations

- enum CyFx3BootSpiSsnLagLead_t {
  CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ZERO_CLK = 0, CY_FX3_BOOT_SPI_SSN_LAG_LEAD_HALF_CLK, CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_CLK, CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_HALF_CLK,
  CY_FX3_BOOT_SPI_NUM_SSN_LAG_LEAD }

  Enumeration defining SSN lead and lag times with respect to SCK.
- enum CyFx3BootSpiSsnCtrl_t {
  CY_FX3_BOOT_SPI_SSN_CTRL_FW = 0, CY_FX3_BOOT_SPI_SSN_CTRL_HW_END_OF_XFER, CY_FX3_BOOT_SPI_SSN_CTRL_HW_EACH_WORD, CY_FX3_BOOT_SPI_SSN_CTRL_HW_CPHA_BASED,
  CY_FX3_BOOT_SPI_SSN_CTRL_NONE, CY_FX3_BOOT_SPI_NUM_SSN_CTRL }

  List of ways in which the SPI Slave Select (SSN) line can be controlled.

Functions

- CyFx3BootErrorCode_t CyFx3BootSpiInit (void)

  Starts the SPI interface block on the device.
- void CyFx3BootSpiDeInit (void)

  Stops the SPI block.
- CyFx3BootErrorCode_t CyFx3BootSpiSetConfig (CyFx3BootSpiConfig_t *config)

  Configures SPI interface parameters.
- void CyFx3BootSpiSetSsnLine (CyBool_t isHigh)

  Assert / Deassert the SSN Line.

- CyFx3BootErrorCode_t CyFx3BootSpiTransmitWords (uint8_t *data, uint32_t byteCount)

  Transmits data word by word over the SPI interface.
- CyFx3BootErrorCode_t CyFx3BootSpiReceiveWords (uint8_t *data, uint32_t byteCount)

  Receives data word by word over the SPI interface.
- void CyFx3BootSpiSetBlockXfer (uint32_t txSize, uint32_t rxSize)

  Enables DMA data transfers through the SPI interface.
- void CyFx3BootSpiDisableBlockXfer (void)

  Disable DMA data transfers through the SPI interface.
- CyFx3BootErrorCode_t CyFx3BootSpiDmaXferData (CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout)

  This function is used to setup a DMA transfer from CPU to SPI or vice versa.

5.5.1 Detailed Description

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. This file defines the data structures and APIs that enable SPI slave access from the FX3 boot API library.

5.5.2 Typedef Documentation

5.5.2.1 typedef struct CyFx3BootSpiConfig_t CyFx3BootSpiConfig_t

Structure defining the configuration of SPI interface.

Description

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

See Also

CyFx3BootSpiSetConfig
CyFx3BootSpiSsnCtrl_t
CyFx3BootSpiSsnLagLead_t

5.5.2.2 typedef enum CyFx3BootSpiSsnCtrl_t CyFx3BootSpiSsnCtrl_t

List of ways in which the SPI Slave Select (SSN) line can be controlled.

Description

The SPI Slave Select (SSN) signal is a per-slave signal; and multiple SSN signals may be needed in the case where FX3 needs to talk to multiple SPI slaves. To satisfy this need, the FX3 device supports multiple ways in which the SSN signal can be controlled during a SPI data transfer.

The default SSN signal on FX3 can be automatically controlled by the FX3 hardware in sync with the SPI clock (SCK). Another possibility is to have the SSN signal controlled as a GPIO by the SPI transfer APIs. This functionality is also supported only for the default SSN signal.

A third option is for the firmware application to directly assert/de-assert the SSN signal through the GPIO APIs. This mode of operation can be used with any FX3 GPIO.

See Also

CyFx3BootSpiConfig_t
CyFx3BootSpiSetConfig

5.5.2.3 typedef enum CyFx3BootSpiSsnLagLead_t CyFx3BootSpiSsnLagLead_t

Enumeration defining SSN lead and lag times with respect to SCK.

Description

The Slave Select (SSN) signal needs to lead the SCK at the beginning of a SPI transaction and lag the SCK at the end of the transfer. When the default SSN signal on the FX3 is used, the hardware provides various modes of automatically controlling the SSN assertion. This enumerated type lists the various SSN timings that are supported by the FX3 hardware.

See Also

CyFx3BootSpiConfig_t
CyFx3BootSpiSetConfig

5.5.3 Enumeration Type Documentation

5.5.3.1 enum CyFx3BootSpiSsnCtrl_t

List of ways in which the SPI Slave Select (SSN) line can be controlled.

Description

The SPI Slave Select (SSN) signal is a per-slave signal; and multiple SSN signals may be needed in the case where FX3 needs to talk to multiple SPI slaves. To satisfy this need, the FX3 device supports multiple ways in which the SSN signal can be controlled during a SPI data transfer.

The default SSN signal on FX3 can be automatically controlled by the FX3 hardware in sync with the SPI clock (SCK). Another possibility is to have the SSN signal controlled as a GPIO by the SPI transfer APIs. This functionality is also supported only for the default SSN signal.

A third option is for the firmware application to directly assert/de-assert the SSN signal through the GPIO APIs. This mode of operation can be used with any FX3 GPIO.

See Also

CyFx3BootSpiConfig_t
CyFx3BootSpiSetConfig

Enumerator

CY_FX3_BOOT_SPI_SSN_CTRL_FW SSN is controlled by the API and is not synchronized to clock boundaries. It is asserted at the beginning of a transfer, and is de-asserted at the end of transfer.

CY_FX3_BOOT_SPI_SSN_CTRL_HW_END_OF_XFER SSN is controlled by hardware and is toggled in sync with clock. The SSN is asserted at the beginning of a transfer, and is de-asserted at the end of a transfer or when no data is available to transmit (underrun).

CY_FX3_BOOT_SPI_SSN_CTRL_HW_EACH_WORD SSN is controlled by hardware and is toggled in sync with clock. The SSN is asserted at the beginning of transfer of every word, and de-asserted at the end of the transfer of that word.

CY_FX3_BOOT_SPI_SSN_CTRL_HW_CPHA_BASED If CPHA is 0, the SSN control is per word; and if CPHA is 1, then the SSN control is per transfer.

CY_FX3_BOOT_SPI_SSN_CTRL_NONE SSN control is done externally. The SSN lines are selected by the application and are ignored by the hardware / API.

CY_FX3_BOOT_SPI_NUM_SSN_CTRL Number of enumerations. Should not be used.

5.5.3.2 enum CyFx3BootSpiSsnLagLead_t

Enumeration defining SSN lead and lag times with respect to SCK.

Description

The Slave Select (SSN) signal needs to lead the SCK at the beginning of a SPI transaction and lag the SCK at the end of the transfer. When the default SSN signal on the FX3 is used, the hardware provides various modes of automatically controlling the SSN assertion. This enumerated type lists the various SSN timings that are supported by the FX3 hardware.

See Also

CyFx3BootSpiConfig_t
CyFx3BootSpiSetConfig

Enumerator

CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ZERO_CLK SSN will be asserted in sync with SCK.

CY_FX3_BOOT_SPI_SSN_LAG_LEAD_HALF_CLK SSN leads / lags SCK by a half clock cycle.

CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_CLK SSN leads / lags SCK by one clock cycle.

CY_FX3_BOOT_SPI_SSN_LAG_LEAD_ONE_HALF_CLK SSN leads / lags SCK by one and half clock cycles.

CY_FX3_BOOT_SPI_NUM_SSN_LAG_LEAD Number of enumerations. Should not be used.

5.5.4 Function Documentation

5.5.4.1 void CyFx3BootSpiDeInit (void)

Stops the SPI block.

Description

This function disables and powers off the SPI interface. This function can be used to shut off the interface to save power when it is not in use.

Return value

None

See Also

CyFx3BootSpiInit

5.5.4.2 void CyFx3BootSpiDisableBlockXfer (void)

Disable DMA data transfers through the SPI interface.

Description

This function disables DMA data transfers through the SPI interface, so that register mode transfers can be performed again.

Return value

None

See Also

CyFx3BootSpiSetBlockXfer

5.5.4.3 CyFx3BootErrorCode_t CyFx3BootSpiDmaXferData (CyBool_t *isRead*, uint32_t *address*, uint32_t *length*, uint32_t *timeout*)

This function is used to setup a DMA transfer from CPU to SPI or vice versa.

Description

This function is used to read/write data from/to a SPI slave in DMA mode. The contents of a single data buffer can be transferred through a one-shot DMA pipe that is configured by this API call.

Note

The CyFx3BootSpiSetBlockXfer API needs to be called to configure the SPI block for DMA bound data transfers.

Return value

CY_FX3_BOOT_SUCCESS - if the data transfer is successful.

CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR - if the address specified is not in the SYSMEM area.

CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error.

CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out.

See Also

CyFx3BootSpiDisableBlockXfer
CyFx3BootSpiSetBlockXfer

Parameters

isRead	Transfer direction - isRead=CyTrue for read operations; and isRead=CyFalse for write operations.
address	Address of the buffer from/to which data is to be transferred.
length	Length of the data to be transferred.
timeout	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to compulsorily wait for end of transfer.

5.5.4.4 CyFx3BootErrorCode_t CyFx3BootSpiInit (void)

Starts the SPI interface block on the device.

Description

This function powers up the SPI interface block on the device and is expected to be the first SPI API function that is called by the application.

Return value

CY_FX3_BOOT_SUCCESS - if the SPI block was successfully initialized.

See Also

CyFx3BootSpiDeInit
CyFx3BootSpiSetConfig

5.5.4.5 CyFx3BootErrorCode_t CyFx3BootSpiReceiveWords (uint8_t * *data*, uint32_t *byteCount*)

Receives data word by word over the SPI interface.

Description

Receives data from the SPI slave in word-by-word register mode of transfer. This can be used only when a DMA transfer on the SPI block has not been started using the CyFx3BootSpiSetBlockXfer API.

Return value

CY_FX3_BOOT_SUCCESS - when the ReceiveWords is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - when the data pointer is NULL.

CY_FX3_BOOT_ERROR_XFER_FAILURE - when the SPI block encounters an error during the transfer.

CY_FX3_BOOT_ERROR_TIMEOUT - when the ReceiveWords times out.

See Also

CyFx3BootSpiTransmitWords

Parameters

data	Destination buffer pointer.
byteCount	Amount of data to be received (in bytes). This needs to be a multiple of the word length after alignment to the next byte value.

5.5.4.6 void CyFx3BootSpiSetBlockXfer (uint32_t *txSize*, uint32_t *rxSize*)

Enables DMA data transfers through the SPI interface.

Description

This function switches the SPI block to DMA mode, and initiates the desired read/write transfers.

If the txSize parameter is non-zero, then TX is enabled; and if rxSize parameter is non-zero, then RX is enabled. If both TX and RX are enabled, transfers can only happen while both the SPI producer and SPI consumer sockets are ready for data transfer.

If the receive count is less than the transmit count, the data transmit continues after the scheduled reception is complete. However, if the transmit count is lesser, data reception is stalled at the end of the transmit operation. The SPI transmit transfer needs to disabled before the receive can proceed.

The CyFx3BootSpiDmaXferData API needs to be used to set up the DMA data path to do the actual data transfer. A call to SetBlockXfer has to be followed by a call to DisableBlockXfer, before the SPI block can be used in Register mode.

Return value

None

See Also

CyFx3BootSpiDisableBlockXfer
CyFx3BootSpiDmaXferData

Parameters

txSize	Number of words to be transmitted.
rxSize	Number of words to be received.

5.5.4.7 CyFx3BootErrorCode_t CyFx3BootSpiSetConfig (CyFx3BootSpiConfig_t * *config*)

Configures SPI interface parameters.

Description

This function is used to configure the SPI master interface based on the desired settings to talk to the desired slave. This function can be called repeatedly to change the settings if different settings are to be used to communicate

with different slave devices.

This API resets the TX/RX FIFOs associated with the SPI block, and will result in the loss of any data that is in the FIFOs.

Return value

CY_FX3_BOOT_SUCCESS - when the SetConfig is successful.

CY_FX3_BOOT_ERROR_NOT_STARTED - if the SPI block has not been initialized.

CY_FX3_BOOT_ERROR_NULL_POINTER - when the config pointer is NULL.

CY_FX3_BOOT_ERROR_TIMEOUT - when the operation times out.

See Also

CyFx3BootSpiConfig_t

Parameters

config	Pointer to the SPI config structure

5.5.4.8 void CyFx3BootSpiSetSsnLine (CyBool_t *isHigh*)

Assert / Deassert the SSN Line.

Description

Asserts/de-asserts the SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

Return value

None

See Also

CyFx3BootSpiSsnCtrl_t

Parameters

isHigh	CyFalse: Pull down the SSN line, CyTrue: Pull up the SSN line.

5.5.4.9 CyFx3BootErrorCode_t CyFx3BootSpiTransmitWords (uint8_t * *data*, uint32_t *byteCount*)

Transmits data word by word over the SPI interface.

Description

This function is used to transmit data word by word. The function can be called only if there is no active DMA transfer on the bus. If CyFx3BootSpiSetBlockXfer has been called, the CyFx3BootSpiDisableBlockXfer API needs to be called before this API can be used.

Return value

CY_FX3_BOOT_SUCCESS - when the TransmitWords is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - when the data pointer is NULL.

CY_FX3_BOOT_ERROR_XFER_FAILURE - when the SPI block encounters an error during the transfer.

CY_FX3_BOOT_ERROR_TIMEOUT - when the TransmitWords times out.

See Also

CyFx3BootSpiReceiveWords

Parameters

data	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
byteCount	This needs to be a multiple of the word length aligned to the next byte value.

5.6 firmware/boot_fw/include/cyfx3uart.h File Reference

The UART interface manager module is responsible for handling the transfer of data through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

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

Data Structures

- struct CyFx3BootUartConfig_t

  Configuration parameters for the UART interface.

Typedefs

- typedef enum CyFx3BootUartBaudrate_t CyFx3BootUartBaudrate_t

  List of baud rates supported by the UART.
- typedef enum CyFx3BootUartStopBit_t CyFx3BootUartStopBit_t

  List of number of stop bits to be used in UART communication.
- typedef enum CyFx3BootUartParity_t CyFx3BootUartParity_t

  List of parity settings supported by the UART interface.
- typedef struct CyFx3BootUartConfig_t CyFx3BootUartConfig_t

  Configuration parameters for the UART interface.

Enumerations

- enum CyFx3BootUartBaudrate_t { CY_FX3_BOOT_UART_BAUDRATE_4800 = 4800, CY_FX3_BOOT_UART_BAUDRATE_9600 = 9600, CY_FX3_BOOT_UART_BAUDRATE_19200 = 19200, CY_FX3_BOOT_UART_BAUDRATE_38400 = 38400, CY_FX3_BOOT_UART_BAUDRATE_57600 = 57600, CY_FX3_BOOT_UART_BAUDRATE_115200 = 115200 }

  List of baud rates supported by the UART.
- enum CyFx3BootUartStopBit_t { CY_FX3_BOOT_UART_ONE_STOP_BIT = 1, CY_FX3_BOOT_UART_TWO_STOP_BIT = 2 }

  List of number of stop bits to be used in UART communication.
- enum CyFx3BootUartParity_t { CY_FX3_BOOT_UART_NO_PARITY = 0, CY_FX3_BOOT_UART_EVEN_PARITY, CY_FX3_BOOT_UART_ODD_PARITY, CY_FX3_BOOT_UART_NUM_PARITY }

  List of parity settings supported by the UART interface.

Functions

- CyFx3BootErrorCode_t CyFx3BootUartInit (void)

  Starts the UART hardware block on the device.
- void CyFx3BootUartDeInit (void)

  Stops the UART hardware block.
- CyFx3BootErrorCode_t CyFx3BootUartSetConfig (CyFx3BootUartConfig_t ∗config)

  Sets the UART interface parameters.
- void CyFx3BootUartTxSetBlockXfer (uint32_t txSize)

  Sets the UART transmit data count in the case of DMA mode operation.
- void CyFx3BootUartRxSetBlockXfer (uint32_t rxSize)

  Sets the number of bytes to be received by the UART in DMA mode.
- uint32_t CyFx3BootUartTransmitBytes (uint8_t ∗data_p, uint32_t count)

  Transmits data through the UART interface on a byte by byte basis.
- uint32_t CyFx3BootUartReceiveBytes (uint8_t ∗data_p, uint32_t count)

  Receives data from the UART interface on a byte by byte basis.
- CyFx3BootErrorCode_t CyFx3BootUartDmaXferData (CyBool_t isRead, uint32_t address, uint32_t length, uint32_t timeout)

  This function is used to setup a DMA from CPU to UART or vice versa.

5.6.1 Detailed Description

The UART interface manager module is responsible for handling the transfer of data through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

5.6.2 Typedef Documentation

5.6.2.1 typedef enum CyFx3BootUartBaudrate_t CyFx3BootUartBaudrate_t

List of baud rates supported by the UART.

Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware.

See Also

CyFx3BootUartConfig_t

5.6.2.2 typedef struct CyFx3BootUartConfig_t CyFx3BootUartConfig_t

Configuration parameters for the UART interface.

Description

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

See Also

CyFx3BootUartBaudrate_t
CyFx3BootUartStopBit_t
CyFx3BootUartParity_t
CyFx3BootUartSetConfig

5.6.2.3 typedef enum CyFx3BootUartParity_t CyFx3BootUartParity_t

List of parity settings supported by the UART interface.

Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

See Also

CyFx3BootUartConfig_t

5.6.2.4 typedef enum CyFx3BootUartStopBit_t CyFx3BootUartStopBit_t

List of number of stop bits to be used in UART communication.

Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have. Only 1 and 2 are supported on the FX3 device.

See Also

CyFx3BootUartConfig_t

5.6.3 Enumeration Type Documentation

5.6.3.1 enum CyFx3BootUartBaudrate_t

List of baud rates supported by the UART.

Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware.

See Also

CyFx3BootUartConfig_t

Enumerator

CY_FX3_BOOT_UART_BAUDRATE_4800 4800 baud.

CY_FX3_BOOT_UART_BAUDRATE_9600 9600 baud.

CY_FX3_BOOT_UART_BAUDRATE_19200 19200 baud.

CY_FX3_BOOT_UART_BAUDRATE_38400 38400 baud.

CY_FX3_BOOT_UART_BAUDRATE_57600 57600 baud.

CY_FX3_BOOT_UART_BAUDRATE_115200 115200 baud.

5.6.3.2 enum CyFx3BootUartParity_t

List of parity settings supported by the UART interface.

Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

See Also

CyFx3BootUartConfig_t

Enumerator

CY_FX3_BOOT_UART_NO_PARITY No parity bits.

CY_FX3_BOOT_UART_EVEN_PARITY Even parity.

CY_FX3_BOOT_UART_ODD_PARITY Odd parity.

CY_FX3_BOOT_UART_NUM_PARITY Number of parity enumerations.

5.6.3.3 enum CyFx3BootUartStopBit_t

List of number of stop bits to be used in UART communication.

Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have. Only 1 and 2 are supported on the FX3 device.

See Also

CyFx3BootUartConfig_t

Enumerator

CY_FX3_BOOT_UART_ONE_STOP_BIT 1 stop bit

CY_FX3_BOOT_UART_TWO_STOP_BIT 2 stop bit

5.6.4 Function Documentation

5.6.4.1 void CyFx3BootUartDeInit (void)

Stops the UART hardware block.

Description

This function disables and powers off the UART hardware block on the device.

Return value

None

See Also

CyFx3BootUartInit

5.6.4.2 CyFx3BootErrorCode_t CyFx3BootUartDmaXferData (CyBool_t *isRead*, uint32_t *address*, uint32_t *length*, uint32_t *timeout*)

This function is used to setup a DMA from CPU to UART or vice versa.

Description

This function is used to read/write a block of data through the UART interface in DMA mode. This is a blocking call which returns only when the transfer is complete, or the specified timeout duration has elapsed.

This API can only be called if the UART is setup for DMA mode of operation.

Return value

CY_FX3_BOOT_SUCCESS - if data transfer is successful.

CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR - if the address specified is not in the SYSMEM area.

CY_FX3_BOOT_ERROR_XFER_FAILURE - if the data transfer encountered any error.

CY_FX3_BOOT_ERROR_TIMEOUT - if the data transfer times out.

See Also

CyFx3BootUartTxSetBlockXfer
CyFx3BootUartRxSetBlockXfer

Parameters

isRead	Transfer direction - isRead=CyTrue for read operations; and isRead=CyFalse for write operations.
address	Address of the buffer from/to which data is to be transferred.
length	Length of the data to be transferred.
timeout	Timeout duration in multiples of 10 us. Can be set to CY_FX3_BOOT_WAIT_FOREVER to compulsorily wait for end of transfer.

5.6.4.3 CyFx3BootErrorCode_t CyFx3BootUartInit (void)

Starts the UART hardware block on the device.

Description

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

Return value

CY_FX3_BOOT_SUCCESS - when the UART block is successfully initialized.

See Also

CyFx3BootUartDeInit
CyFx3BootUartSetConfig

5.6.4.4 uint32_t CyFx3BootUartReceiveBytes (uint8_t * *data_p*, uint32_t *count*)

Receives data from the UART interface on a byte by byte basis.

Description

This function is used to read "count" number of bytes from the UART register interface. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

Return value

Amount of data actually transferred. A return of zero may indicate a transfer error.

See Also

CyFx3BootUartTransmitBytes

Parameters

data_p	Pointer to location where the data read is to be placed.
count	Number of bytes to be received.

5.6.4.5 void CyFx3BootUartRxSetBlockXfer (uint32_t *rxSize*)

Sets the number of bytes to be received by the UART in DMA mode.

Description

This function sets the size of the desired data reception through the UART. This function is to be used when the UART is configured for DMA mode of transfer.

Return value

None

See Also

CyFx3BootUartTxSetBlockXfer

Parameters

rxSize	Desired transfer size.

5.6.4.6 CyFx3BootErrorCode_t CyFx3BootUartSetConfig (CyFx3BootUartConfig_t * *config*)

Sets the UART interface parameters.

Description

This function configures the UART block with the desired user parameters such as transfer mode, baud rate etc. This function should be called repeatedly to make any change to the set of configuration parameters. This can be called on the fly repetitively without calling CyFx3BootUartInit. But this will reset the FIFO and hence the data in pipe will be lost.

Return value

CY_FX3_BOOT_SUCCESS - if the configuration was set successfully.

CY_FX3_BOOT_ERROR_NOT_STARTED - if the UART block has not been initialized.

CY_FX3_BOOT_ERROR_NULL_POINTER - if a NULL pointer is passed.

See Also

CyFx3BootUartConfig_t

Parameters

config	Pointer to structure containing config information.

5.6.4.7 uint32_t CyFx3BootUartTransmitBytes (uint8_t * *data_p*, uint32_t *count*)

Transmits data through the UART interface on a byte by byte basis.

Description

This function is used to transfer "count" number of bytes out through the UART register interface. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

Return value

Amount of data actually transferred. A return of zero may indicate a transfer error.

See Also

CyFx3BootUartReceiveBytes

Parameters

data_p	Pointer to the data to be transferred.
count	Number of bytes to be transferred.

5.6.4.8 void CyFx3BootUartTxSetBlockXfer (uint32_t *txSize*)

Sets the UART transmit data count in the case of DMA mode operation.

Description

This function sets the size of the desired data transmission through the UART. This function is to be used when the UART is configured for DMA mode of transfer.

Return value

None

See Also

CyFx3BootUartRxSetBlockXfer

Parameters

txSize	Desired transfer size.

5.7 firmware/boot_fw/include/cyfx3usb.h File Reference

The FX3 boot library implements a USB driver that supports the device mode of USB operation (no host or OTG support). This file defines the data structures and APIs provided by the USB driver to control the USB operation.

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

Data Structures

- struct CyU3PUsbDescrPtrs

  Pointer to the various descriptors.
- struct CyFx3BootUsbEp0Pkt_t

  USB Setup Packet data structure.
- struct CyFx3BootUsbEpConfig_t

  Endpoint configuration information.

Macros

- #define CY_FX3_USB_MAX_STRING_DESC_INDEX (16)

  Number of string descriptors that are supported by the FX3 booter.

Typedefs

- typedef enum CyFx3BootUsbSpeed_t CyFx3BootUsbSpeed_t

  List of supported USB connection speeds.
- typedef enum CyFx3BootUsbEventType_t CyFx3BootUsbEventType_t

  Enumeration of USB event types.
- typedef enum CyFx3BootUsbEpType_t CyFx3BootUsbEpType_t

  Enumeration of the endpoint types.
- typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t

  Virtual descriptor types to be used to set descriptors.
- typedef enum CyU3PUsbDescType CyU3PUsbDescType

  Enumeration of USB descriptor types.
- typedef struct CyU3PUsbDescrPtrs CyU3PUsbDescrPtrs

  Pointer to the various descriptors.
- typedef struct CyFx3BootUsbEp0Pkt_t CyFx3BootUsbEp0Pkt_t

  USB Setup Packet data structure.
- typedef struct CyFx3BootUsbEpConfig_t CyFx3BootUsbEpConfig_t

  Endpoint configuration information.
- typedef void(∗ CyFx3BootUSBEventCb_t)(CyFx3BootUsbEventType_t event)

  USB Events notification callback.
- typedef void(∗ CyFx3BootUSBSetupCb_t)(uint32_t setupDat0, uint32_t setupDat1)

  USB setup request handler type.

Enumerations

- enum CyFx3BootUsbSpeed_t { CY_FX3_BOOT_NOT_CONNECTED = 0x00, CY_FX3_BOOT_FULL_SPEED, CY_FX3_BOOT_HIGH_SPEED, CY_FX3_BOOT_SUPER_SPEED }

  List of supported USB connection speeds.
- enum CyFx3BootUsbEventType_t { CY_FX3_BOOT_USB_CONNECT = 0x00, CY_FX3_BOOT_USB_DISCONNECT, CY_FX3_BOOT_USB_RESET, CY_FX3_BOOT_USB_SUSPEND, CY_FX3_BOOT_USB_RESUME, CY_FX3_BOOT_USB_IN_SS_DISCONNECT, CY_FX3_BOOT_USB_COMPLIANCE }

  Enumeration of USB event types.
- enum CyFx3BootUsbEpType_t { CY_FX3_BOOT_USB_EP_CONTROL = 0, CY_FX3_BOOT_USB_EP_ISO, CY_FX3_BOOT_USB_EP_BULK, CY_FX3_BOOT_USB_EP_INTR }

  Enumeration of the endpoint types.
- enum CyU3PUSBSetDescType_t { CY_U3P_USB_SET_SS_DEVICE_DESCR, CY_U3P_USB_SET_HS_DEVICE_DESCR, CY_U3P_USB_SET_DEVQUAL_DESCR, CY_U3P_USB_SET_FS_CONFIG_DESCR, CY_U3P_USB_SET_HS_CONFIG_DESCR, CY_U3P_USB_SET_STRING_DESCR, CY_U3P_USB_SET_SS_CONFIG_DESCR, CY_U3P_USB_SET_SS_BOS_DESCR, CY_U3P_USB_SET_SS_DEVICE_DESCR, CY_U3P_USB_SET_HS_DEVICE_DESCR, CY_U3P_USB_SET_DEVQUAL_DESCR, CY_U3P_USB_SET_FS_CONFIG_DESCR, CY_U3P_USB_SET_HS_CONFIG_DESCR, CY_U3P_USB_SET_STRING_DESCR, CY_U3P_USB_SET_SS_CONFIG_DESCR, CY_U3P_USB_SET_SS_BOS_DESCR, CY_U3P_USB_SET_OTG_DESCR }

  Virtual descriptor types to be used to set descriptors.

- enum 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_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, 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 }

  Enumeration of USB descriptor types.

Functions

- CyFx3BootErrorCode_t CyFx3BootUsbStart (CyBool_t noReEnum, CyFx3BootUSBEventCb_t cb)

  Start the USB driver.
- CyFx3BootErrorCode_t CyFx3BootUsbSetDesc (CyU3PUSBSetDescType_t descType, uint8_t desc_index, uint8_t *desc)

  Register a USB descriptor with the booter.
- void CyFx3BootUsbConnect (CyBool_t connect, CyBool_t ssEnable)

  Enable/Disable USB connection.
- void CyFx3BootRegisterSetupCallback (CyFx3BootUSBSetupCb_t callback)

  Function to register a USB setup request handler.
- CyFx3BootErrorCode_t CyFx3BootUsbGetEpCfg (uint8_t ep, CyBool_t *isNak, CyBool_t *isStall)

  Retrieve the current state of the specified endpoint.
- CyFx3BootErrorCode_t CyFx3BootUsbSetClrFeature (uint32_t sc, CyBool_t isConfigured, CyFx3BootUsbEp0Pkt_t *pEp0)

  Set/Clear Feature USB Request handler.
- CyFx3BootErrorCode_t CyFx3BootUsbDmaXferData (uint8_t epNum, uint32_t address, uint32_t length, uint32_t timeout)

  Function to transfer USB endpoint data using DMA.
- CyFx3BootUsbSpeed_t CyFx3BootUsbGetSpeed (void)

  Get the connection speed at which USB is operating.
- void CyFx3BootUsbStall (uint8_t ep, CyBool_t stall, CyBool_t toggle)

  Set or clear the stall status of an endpoint.
- void CyFx3BootUsbAckSetup (void)

  Complete the status handshake of a USB control request.
- CyU3PUsbDescrPtrs * CyFx3BootUsbGetDesc (void)

  Retrieves the pointer to the USB descriptors.
- CyFx3BootErrorCode_t CyFx3BootUsbSetEpConfig (uint8_t ep, CyFx3BootUsbEpConfig_t *epinfo)

  Configure a USB endpoint's properties.
- void CyFx3BootUsbVBattEnable (CyBool_t enable)

  Configure USB block on FX3 to work off Vbatt power instead of Vbus.
- void CyFx3BootUsbEp0StatusCheck (void)

  Function to check if the status stage of a EP0 transfer is pending.
- void CyFx3BootUsbCheckUsb3Disconnect (void)

  This function checks if the device state is in USB 3.0 disconnect state.
- void CyFx3BootUsbSendCompliancePatterns (void)

This function is used to send the compliance patterns.

- void CyFx3BootUsbLPMDisable (void)

  Disable acceptance of U1/U2 entry requests from USB 3.0 host.

- void CyFx3BootUsbLPMEnable (void)

  Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.

5.7.1 Detailed Description

The FX3 boot library implements a USB driver that supports the device mode of USB operation (no host or OTG support). This file defines the data structures and APIs provided by the USB driver to control the USB operation.

5.7.2 Macro Definition Documentation

5.7.2.1 #define CY_FX3_USB_MAX_STRING_DESC_INDEX (16)

Number of string descriptors that are supported by the FX3 booter.

Description

The FX3 booter is capable of storing pointers to and handling a number of string descriptors. This constant represents the number of strings that the booter is capable of handling and is currently fixed to 16.

5.7.3 Typedef Documentation

5.7.3.1 typedef struct CyFx3BootUsbEp0Pkt_t CyFx3BootUsbEp0Pkt_t

USB Setup Packet data structure.

Description

Control Endpoint setup packet data structure.

5.7.3.2 typedef struct CyFx3BootUsbEpConfig_t CyFx3BootUsbEpConfig_t

Endpoint configuration information.

Description

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

5.7.3.3 typedef void(∗ CyFx3BootUSBEventCb_t)(CyFx3BootUsbEventType_t event)

USB Events notification callback.

Description

Type of callback function that is invoked to notify the USB events. The FX3 booter does the necessary handling of the USB events and the application should NOT block on this function.

5.7.3.4 typedef void(∗ CyFx3BootUSBSetupCb_t)(uint32_t setupDat0,uint32_t setupDat1)

USB setup request handler type.

Description

Type of callback function that is invoked to handle USB setup requests. The booter doesn't handle any of the standard/vendor requests and this handler is expected to handle all such requests.

5.7.3.5 typedef struct CyU3PUsbDescrPtrs CyU3PUsbDescrPtrs

Pointer to the various descriptors.

Description

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

5.7.3.6 typedef enum CyU3PUsbDescType CyU3PUsbDescType

Enumeration of USB descriptor types.

Description

Descriptor types as defined in the USB Specification.

5.7.3.7 typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t

Virtual descriptor types to be used to set descriptors.

Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

5.7.4 Enumeration Type Documentation

5.7.4.1 enum CyFx3BootUsbEpType_t

Enumeration of the endpoint types.

Enumerator

CY_FX3_BOOT_USB_EP_CONTROL Control Endpoint Type

CY_FX3_BOOT_USB_EP_ISO Isochronous Endpoint Type

CY_FX3_BOOT_USB_EP_BULK Bulk Endpoint Type

CY_FX3_BOOT_USB_EP_INTR Interrupt Endpoint Type

5.7.4.2 enum CyFx3BootUsbEventType_t

Enumeration of USB event types.

Enumerator

CY_FX3_BOOT_USB_CONNECT USB Connect Event

CY_FX3_BOOT_USB_DISCONNECT USB DisConnect Event

CY_FX3_BOOT_USB_RESET USB Reset Event

CY_FX3_BOOT_USB_SUSPEND USB Suspend Event

CY_FX3_BOOT_USB_RESUME USB Resume Event

CY_FX3_BOOT_USB_IN_SS_DISCONNECT Event to check if the device is in SS Disconnect State

CY_FX3_BOOT_USB_COMPLIANCE USB Compliance Event

5.7.4.3 enum CyFx3BootUsbSpeed_t

List of supported USB connection speeds.

Enumerator

CY_FX3_BOOT_NOT_CONNECTED USB device not connected.

CY_FX3_BOOT_FULL_SPEED USB full speed.

CY_FX3_BOOT_HIGH_SPEED High speed.

CY_FX3_BOOT_SUPER_SPEED Super speed.

5.7.4.4 enum CyU3PUsbDescType

Enumeration of USB descriptor types.

Description

Descriptor types as defined in the USB Specification.

Enumerator

CY_U3P_USB_DEVICE_DESCR Super Speed Device descr

CY_U3P_USB_CONFIG_DESCR Configuration

CY_U3P_USB_STRING_DESCR String

CY_U3P_USB_INTRFC_DESCR Interface

CY_U3P_USB_ENDPNT_DESCR End Point

CY_U3P_USB_DEVQUAL_DESCR Device Qualifier

CY_U3P_USB_OTHERSPEED_DESCR Other Speed Configuration

CY_U3P_USB_INTRFC_POWER_DESCR Interface power descriptor

CY_U3P_BOS_DESCR BOS descriptor

CY_U3P_DEVICE_CAPB_DESCR Device Capability descriptor

CY_U3P_USB_HID_DESCR HID descriptor

CY_U3P_USB_REPORT_DESCR Report descriptor

CY_U3P_SS_EP_COMPN_DESCR End Point companion descriptor

CY_U3P_USB_DEVICE_DESCR Device descriptor

CY_U3P_USB_CONFIG_DESCR Configuration descriptor

CY_U3P_USB_STRING_DESCR String descriptor

CY_U3P_USB_INTRFC_DESCR Interface descriptor

CY_U3P_USB_ENDPNT_DESCR Endpoint descriptor

CY_U3P_USB_DEVQUAL_DESCR Device Qualifier descriptor

CY_U3P_USB_OTHERSPEED_DESCR Other Speed Configuration descriptor

CY_U3P_USB_INTRFC_POWER_DESCR Interface power descriptor descriptor

CY_U3P_USB_OTG_DESCR OTG descriptor

CY_U3P_BOS_DESCR BOS descriptor

CY_U3P_DEVICE_CAPB_DESCR Device Capability descriptor

CY_U3P_USB_HID_DESCR HID descriptor

CY_U3P_USB_REPORT_DESCR Report descriptor

CY_U3P_SS_EP_COMPN_DESCR Endpoint companion descriptor

5.7.4.5 enum CyU3PUSBSetDescType_t

Virtual descriptor types to be used to set descriptors.

Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

Enumerator

CY_U3P_USB_SET_SS_DEVICE_DESCR SuperSpeed (USB 3.0) device descriptor.

CY_U3P_USB_SET_HS_DEVICE_DESCR USB 2.0 device descriptor.

CY_U3P_USB_SET_DEVQUAL_DESCR Device qualifier descriptor

CY_U3P_USB_SET_FS_CONFIG_DESCR Full Speed configuration descriptor.

CY_U3P_USB_SET_HS_CONFIG_DESCR High Speed configuration descriptor.

CY_U3P_USB_SET_STRING_DESCR String descriptor.

CY_U3P_USB_SET_SS_CONFIG_DESCR SuperSpeed configuration descriptor.

CY_U3P_USB_SET_SS_BOS_DESCR BOS descriptor.

CY_U3P_USB_SET_SS_DEVICE_DESCR Device descriptor for SuperSpeed.

CY_U3P_USB_SET_HS_DEVICE_DESCR Device descriptor for Hi-Speed and Full Speed.

CY_U3P_USB_SET_DEVQUAL_DESCR Device Qualifier descriptor.

CY_U3P_USB_SET_FS_CONFIG_DESCR Configuration descriptor for Full Speed.

CY_U3P_USB_SET_HS_CONFIG_DESCR Configuration descriptor for Hi-Speed.

CY_U3P_USB_SET_STRING_DESCR String Descriptor

CY_U3P_USB_SET_SS_CONFIG_DESCR Configuration descriptor for SuperSpeed.

CY_U3P_USB_SET_SS_BOS_DESCR BOS descriptor.

CY_U3P_USB_SET_OTG_DESCR OTG descriptor.

5.7.5 Function Documentation

5.7.5.1 void CyFx3BootRegisterSetupCallback (CyFx3BootUSBSetupCb_t *callback*)

Function to register a USB setup request handler.

Description

This function is used to register a USB setup request handler with the booter. This setup request handler is expected to handle all the USB standard/vendor requests.

Return value

None

See Also

CyFx3BootUSBSetupCb_t

Parameters

callback	Setup request handler

5.7.5.2 void CyFx3BootUsbAckSetup (void)

Complete the status handshake of a USB control request.

Description

This function is used to complete the status handshake of a USB control request that does not involve any data transfer.

5.7.5.3 void CyFx3BootUsbCheckUsb3Disconnect (void)

This function checks if the device state is in USB 3.0 disconnect state.

Description

This function checks if the device is in USB 3.0 disconnect state. If this is the case then a fallback to USB 2.0 is attempted. The function returns immediately if the device is not in the disconnect state.

As this check cannot be done in the ISR context this has been deferred to the application context.

Return value

None

5.7.5.4 void CyFx3BootUsbConnect (CyBool_t *connect,* CyBool_t *ssEnable*)

Enable/Disable USB connection.

Description

This function is used to enable the USB PHYs and to control the connection to the USB host in that manner.

The connect parameter enables/disables the USB connection.

The ssEnable parameter controls the SS or HS/FS enumeration. If SS enumeration is tried and fails, the device automatically falls back to HS/FS mode.

Return value

None

Parameters

connect	CyFalse - Disable USB Connection; CyTrue - Enable USB Connection
ssEnable	CyFalse - HS/FS Enumeration; CyTrue - SS Enumeration

5.7.5.5 CyFx3BootErrorCode_t CyFx3BootUsbDmaXferData (uint8_t *epNum,* uint32_t *address,* uint32_t *length,* uint32_t *timeout*)

Function to transfer USB endpoint data using DMA.

Description

This function can be used to transfer data on a USB endpoint. This works for the control endpoint as well as for other endpoints. The function does not validate the parameters, and can cause a fatal lock-up if called with the wrong parameters.

Return value

CY_FX3_BOOT_SUCCESS - in case of succesful DMA transfer.

CY_FX3_BOOT_ERROR_INVALID_DMA_ADDR - if the address specified is not in the SYSMEM area.

CY_FX3_BOOT_ERROR_XFER_FAILURE - in case of DMA transfer failure.

CY_FX3_BOOT_ERROR_TIMEOUT - in case of DMA transfer times out.

Parameters

epNum	Endpoint number to/from which to transfer data. The MSB is used to identify whether this is a read or a write transfer.
address	SYSMEM address from/to which data is to be transferred.
length	Length of the transfer in bytes.
timeout	Timeout in multiples of 100 us. Also refer the macros CY_FX3_BOOT_NO_WAIT and CY_FX3_BOOT_WAIT_FOREVER

5.7.5.6 void CyFx3BootUsbEp0StatusCheck (void)

Function to check if the status stage of a EP0 transfer is pending.

Description

This function is used to check if the status stage of a EP0 transfer is pending. In order for the device to handle the power management appropriately the device doesn't go into low power modes while a EP0 request is pending.

As this check cannot be done in the ISR context this has been deferred to the application context. This is applicable when the device is functioning in SuperSpeed only.

Return value

None

5.7.5.7 CyU3PUsbDescrPtrs* CyFx3BootUsbGetDesc (void)

Retrieves the pointer to the USB descriptors.

Description

The booter stores the descriptor pointers that are passed in as part of the CyFx3BootUsbSetDesc() function. This function is used to retrieve pointer of type CyU3PUsbDescrPtrs. This pointer can be used to retrieve the relevant data while handling the standard USB requests.

Return value

Pointer of type CyU3PUsbDescrPtrs on Success.

NULL on Failure

See Also

CyFx3BootUsbSetDesc

5.7.5.8 CyFx3BootErrorCode_t CyFx3BootUsbGetEpCfg (uint8_t *ep*, CyBool_t * *isNak*, CyBool_t * *isStall*)

Retrieve the current state of the specified endpoint.

Description

This function retrieves the current NAK and STALL status of the specified endpoint. The isNak return value will be CyTrue if the endpoint is forced to NAK all requests. The isStall return value will be CyTrue if the endpoint is currently stalled.

Return value

CY_FX3_BOOT_SUCCESS - On Success.

CY_FX3_BOOT_ERROR_NULL_POINTER - If isNak or the isStall parameter is NULL.

Parameters

ep	Endpoint number to query.
isNak	Return parameter which will be filled with the NAK status.
isStall	Return parameter which will be filled with the STALL status.

5.7.5.9 CyFx3BootUsbSpeed_t CyFx3BootUsbGetSpeed (void)

Get the connection speed at which USB is operating.

Description

This function is used to get the operating speed of the active USB connection.

Return value

Current USB connection speed.

5.7.5.10 void CyFx3BootUsbLPMDisable (void)

Disable acceptance of U1/U2 entry requests from USB 3.0 host.

Description

In some cases, accepting U1/U2 entry requests from the USB 3.0 host continuously can limit the performance that can be achieved through the FX3. This API can be used to temporarily disable the acceptance of U1/U2 requests by the FX3 device.

Return value

None

See Also

CyFx3BootUsbLPMEnable

5.7.5.11 void CyFx3BootUsbLPMEnable (void)

Re-enable acceptance of U1/U2 entry requests from USB 3.0 host.

Description

This API is used to re-enable acceptance of U1/U2 entry requests by the FX3 device. It is required that LPM acceptance be kept enabled whenever a new USB connection is started up. This is required for passing USB compliance tests.

Return value

None

See Also

CyFx3BootUsbLPMDisable

5.7.5.12 void CyFx3BootUsbSendCompliancePatterns (void)

This function is used to send the compliance patterns.

Description

This function is used to send the USB 3.0 compliance patterns. If the device's LTSSM state is currently in the compliance state, then compliance patterns are to be sent. This function returns when the LTSSM state is no longer in the compliance state.

As this task cannot be done in the ISR context this has been deferred to the application context.

Return value

None

5.7.5.13 CyFx3BootErrorCode_t CyFx3BootUsbSetClrFeature (uint32_t *sc*, CyBool_t *isConfigured*, CyFx3BootUsbEp0Pkt_t * *pEp0*)

Set/Clear Feature USB Request handler.

Description

This function handles standard SET_FEATURE and CLEAR_FEATURE commands from the USB host. The requests handled include EP_HALT, FUNCTION_SUSPEND, Ux_ENABLE, LTM_ENABLE and TEST_MODE features.

Return value

CY_FX3_BOOT_SUCCESS - on success.

CY_FX3_BOOT_ERROR_FAILURE - on failure.

Parameters

sc	1 - Set feature, 0 - Clear feature.
isConfigured	Whether the device is in configured (SET_CONFIG) state.
pEp0	Pointer to the control request structure.

5.7.5.14 CyFx3BootErrorCode_t CyFx3BootUsbSetDesc (CyU3PUSBSetDescType_t *descType*, uint8_t *desc_index*, uint8_t * *desc*)

Register a USB descriptor with the booter.

Description

This function is used to register a USB descriptor with the booter. The booter is capable of remembering one descriptor each of the various supported types as well as upto 16 different string descriptors.

The booter only stores the descriptor pointers that are passed in to this function, and does not make copies of the descriptors. The caller therefore should not free up these descriptor buffers while the USB booter is active.

Note

If the noReEnum option to the CyFx3BootUsbStart API is used, this API makes a copy of the descriptor to pass to the full firmware application. Since a finite memory space (total space of 3 KB) is available for these copied descriptors; this API will return an error when all of the available memory has been used up.

Return value

CY_FX3_BOOT_SUCCESS - On Success.

CY_FX3_BOOT_ERROR_BAD_ARGUMENT - Bad descriptor index.

CY_FX3_BOOT_ERROR_BAD_DESCRIPTOR_TYPE - Bad descriptor type.

CY_FX3_BOOT_ERROR_MEMORY_ERROR - out of memory when copying descriptors.

Parameters

descType	Type of the descriptor
desc_index	Descriptor index for string descriptors only.
desc	Pointer to the descriptor.

5.7.5.15 CyFx3BootErrorCode_t CyFx3BootUsbSetEpConfig (uint8_t *ep*, CyFx3BootUsbEpConfig_t * *epinfo*)

Configure a USB endpoint's properties.

Description

The FX3 device has 30 user configurable endpoints (1-OUT to 15-OUT and 1-IN to 15-IN) which can be separately

selected and configured with desired properties such as endpoint type and the maximum packet size.

All of these endpoints are kept disabled by default. This function is used to enable and set the properties for a specified endpoint. Separate calls need to be made to enable and configure each endpoint that needs to be used.

Note

Only BULK endpoints are supported by the boot firmware as of now.

Return value

CY_FX3_BOOT_SUCCESS - when the call is successful.

CY_FX3_BOOT_ERROR_NULL_POINTER - when the epinfo pointer is NULL.

CY_FX3_BOOT_ERROR_NOT_SUPPORTED - if eptype is non-bulk endpoint.

Parameters

ep	Endpoint number to configured.
epinfo	EP configuration information

5.7.5.16 void CyFx3BootUsbStall (uint8_t *ep,* CyBool_t *stall,* CyBool_t *toggle*)

Set or clear the stall status of an endpoint.

Description

This function is to set or clear the stall status of a given endpoint. This function is to be used in response to SET_FEATURE and CLEAR_FEATURE requests from the host as well as for interface specific error handling. When the stall condition is being cleared, the data toggles for the endpoint can also be cleared. While an option is provided to leave the data toggles unmodified, this should only be used under specific conditions as recommended by Cypress.

Return value

None

Parameters

ep	Endpoint number to be modified.
stall	CyTrue: Set the stall condition, CyFalse: Clear the stall
toggle	CyTrue: Clear the data toggles in a Clear Stall call

5.7.5.17 CyFx3BootErrorCode_t CyFx3BootUsbStart (CyBool_t *noReEnum,* CyFx3BootUSBEventCb_t *cb*)

Start the USB driver.

Description

This function is used to start the USB driver of the booter. This function powers on the USB block on the FX3 device, if it is not already on. The application can register for the USB events of type CyFx3BootUSBEventCb_t.

Return value

CY_FX3_BOOT_ERROR_NO_REENUM_REQUIRED - indicates that the USB block has been left on when transferring control from the main FX3 application to the boot firmware. CY_FX3_BOOT_SUCCESS - USB block and driver have been started up.

Parameters

noReEnum	CyTrue: Do not re-enumerate if the USB connection is active; CyFalse: Force re-enumeration by resetting the USB block.
cb	USB Event handler.

5.7.5.18 void CyFx3BootUsbVBattEnable (CyBool_t *enable*)

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

Description

The USB block on the FX3 device can be configured to work off Vbus power or Vbatt power, with the Vbus power being the default setting. This function is used to enable/disable the Vbatt power input to the USB block.

Note

This call is expected to be done prior to the CyFx3BootUsbConnect.

Return value

None

Parameters

enable	CyTrue: Work off VBatt, CyFalse: Do not work off VBatt

5.8 firmware/boot_fw/include/cyfx3utils.h File Reference

This file provides some generic utility functions for the use of the FX3 boot library.

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

Functions

- void CyFx3BootBusyWait (uint16_t usWait)

  Delay subroutine.

5.8.1 Detailed Description

This file provides some generic utility functions for the use of the FX3 boot library.

5.8.2 Function Documentation

5.8.2.1 void CyFx3BootBusyWait (uint16_t *usWait*)

Delay subroutine.

Description

This function provides delays in multiple of microseconds. The delay is provided by a busy execution loop, which means that the CPU cannot perform any other operations while this code is being executed.

Parameters

usWait	Delay duration in micro-seconds.

5.9 firmware/u3p_firmware/inc/cyfx3_api.h File Reference

Defines the APIs provided by the FX3 core library.

```
#include <cyu3types.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PIoMatrixConfig_t

  IO matrix configuration parameters.

Typedefs

- typedef enum CyU3PPartNumber_t CyU3PPartNumber_t

  Enumeration of EZ-USB FX3 part numbers.
- typedef enum CyU3PIoMatrixLppMode_t CyU3PIoMatrixLppMode_t

  Defines the enumerations for LPP IO line configurations.
- typedef enum CyU3PSPortMode_t CyU3PSPortMode_t

  Define the various operating modes for the storage ports on the FX3S device.
- typedef struct CyU3PIoMatrixConfig_t CyU3PIoMatrixConfig_t

  IO matrix configuration parameters.

Enumerations

- enum CyU3PPartNumber_t { CYPART_USB3014 = 0, CYPART_USB3012, CYPART_USB3013, CYPART_USB3011, CYPART_USB3035, CYPART_USB3034, CYPART_USB3033, CYPART_USB3032, CYPART_USB3031, CYPART_USB2035, CYPART_USB2034, CYPART_USB2033, CYPART_USB2032, CYPART_USB2031, CYPART_USB3025, CYPART_USB3024, CYPART_USB3023, CYPART_USB3021, CYPART_USB2025, CYPART_USB2024, CYPART_USB2023, CYPART_USB3061, CYPART_USB3062, CYPART_USB3063, CYPART_USB3064, CYPART_USB3065, CYPART_USB2064, CYPART_LASTDEV }

  Enumeration of EZ-USB FX3 part numbers.
- 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, CY_U3P_IO_MATRIX_LPP_NONE }

  Defines the enumerations for LPP IO line configurations.
- enum CyU3PSPortMode_t { CY_U3P_SPORT_INACTIVE = 0, CY_U3P_SPORT_4BIT, CY_U3P_SPORT_8BIT }

  Define the various operating modes for the storage ports on the FX3S device.

Functions

- void CyFx3DevInitPageTables (void)

  Setup the default page tables for the FX3 device.
- void CyFx3DevClearSwInterrupt (void)

  Clear a FX3 device specific software interrupt.
- CyU3PPartNumber_t CyFx3DevIdentifyPart (void)

  Get the current FX3/FX3S device part number.
- CyBool_t CyFx3DevIsUsb3Supported (void)

  Check if the part being used supports the USB 3.0 peripheral functionality.

- CyBool_t CyFx3DevIsOtgSupported (void)

  Check if the part being used supports the USB OTG and USB 2.0 host functionality.
- CyBool_t CyFx3DevIsRam512Supported (void)

  Check if the part being used supports 512 KB of System RAM.
- CyBool_t CyFx3DevIsGpifSupported (void)

  Check if the part supports the PIB/GPIF II interface.
- CyBool_t CyFx3DevIsGpif32Supported (void)

  Check if the part supports a 32 bit wide GPIF II data bus.
- CyBool_t CyFx3DevIsSib0Supported (void)

  Check if the part supports the S0 storage port.
- CyBool_t CyFx3DevIsSib1Supported (void)

  Check if the part supports the S1 storage port.
- CyBool_t CyFx3DevIsI2sSupported (void)

  Check if the part supports the I2S interface.
- CyBool_t CyFx3DevIsMipicsiSupported (void)

  Check if the part supports the MIPI CSI interface.
- uint8_t CyFx3DevGetMipiLaneCount (void)

  Get the number of CSI lanes supported by the CX3 part in use.
- CyBool_t CyFx3DevIsGpifConfigurable (void)

  Check if the part supports a configurable GPIF interface.
- CyBool_t CyFx3DevIOIsI2cConfigured (void)

  Check if I2C is enabled in the current IO configuration.
- CyBool_t CyFx3DevIOIsUartConfigured (void)

  Check if UART is enabled in the current IO configuration.
- CyBool_t CyFx3DevIOIsI2sConfigured (void)

  Check if I2S is enabled in the current IO configuration.
- CyBool_t CyFx3DevIOIsSpiConfigured (void)

  Check if SPI is enabled in the current IO configuration.
- CyBool_t CyFx3DevIOIsSib0Configured (void)

  Check if the SIB0 port is configured.
- CyBool_t CyFx3DevIOIsSib1Configured (void)

  Check if the SIB1 port is configured.
- CyBool_t CyFx3DevIOIsSib8BitWide (uint8_t portId)

  Check if the specified SIB port is configured for 8 bit operation.
- CyBool_t CyFx3DevIOIsGpio (uint32_t gpioId, CyBool_t isSimple)

  Check if the specified GPIO is configured as a GPIO.
- void CyFx3DevIOSelectGpio (uint8_t gpioId, CyBool_t enable, CyBool_t isSimple)

  Control the override of a IO pin as a simple or complex GPIO.
- CyBool_t CyFx3DevIOConfigure (CyU3PIoMatrixConfig_t *cfg_p)

  Configure the IO matrix on the device.
- void CyFx3UsbPowerOn (void)

  Power on the USB block on the FX3/FX3S device.
- void CyFx3UsbWritePhyReg (uint16_t phyAddr, uint16_t phyVal)

  Write to a USB 3.0 PHY Control Register.
- void CyFx3Usb2PhySetup (void)

  Configure the USB 2.0 PHY on FX3.
- void CyFx3Usb3LnkSetup (void)

  Configure the USB 3.0 LINK on FX3.
- void CyFx3Usb3SendTP (uint32_t *tpData)

  Send a USB 3.0 Transaction Packet.
- void CyFx3UsbDmaPrefetchEnable (CyBool_t streamEnable)

Enable data prefetch from the DMA sockets on the USB egress (IN) data path.

- void CyFx3Usb3LnkRelaxHpTimeout (void)

  Relax the USB 3.0 HP ACK Timeout period.
- void CyFx3PibPowerOn (void)

  Power on the PIB and GPIF II blocks on the FX3/FX3S device.
- void CyFx3PibPowerOff (void)

  Power off the PIB and GPIF II blocks on the FX3/FX3S device.
- void CyFx3PibDllEnable (void)

  Enable the DLL in the PIB block.
- uint16_t CyFx3PibGetDllStatus (void)

  Get the current status of the PIB DLL.
- void CyFx3SibPowerOn (void)

  Power the storage interface block on the FX3S/SD3 device on.
- void CyFx3SibPowerOff (void)

  Power the storage interface block on the FX3S/SD3 device off.

5.9.1 Detailed Description

Defines the APIs provided by the FX3 core library. **Description**

The FX3 Core library provides a minimal set of API which operate at the device level, and are restricted for IP protection. This header file defines the set of APIs provided by the core library.

5.9.2 Typedef Documentation

5.9.2.1 typedef struct CyU3PIoMatrixConfig_t CyU3PIoMatrixConfig_t

IO matrix configuration parameters.

Description

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

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

Note

A common structure including the storage port configuration is used for both FX3 and FX3S devices, in order to maintain a common API interface. The s0Mode and s1Mode fields should be set to CY_U3P_SPORT_INACTIVE when using the EZ-USB FX3 device.

See Also

CyU3PIoMatrixLppMode_t
CyU3PSPortMode_t
CyU3PDeviceConfigureIOMatrix

5.9.2.2 typedef enum CyU3PIoMatrixLppMode_t CyU3PIoMatrixLppMode_t

Defines the enumerations for LPP IO line configurations.

Description

Most of the IO pins on the FX3 device are multi-purpose with the specific configuration being selected at system start-up. This enumeration lists the various IO operating modes relating to Serial peripheral interfaces.

The I2C interface is always available. If the GPIF data bus is configured as 32-bit wide, only one of the SPI, UART and I2S interfaces are available. In this case, the configuration chosen should be CY_U3P_IO_MATRIX_LPP_DEFAULT.

If the GPIF data bus is 8, 16 or 24 bits wide; it is possible to use all of the SPI, UART and I2S interfaces. However, the peripheral pins can be relocated in this case. Refer to the FX3 datasheet for more details and choose the desired configuration accordingly.

In the case of the FX3S device, none of the UART, SPI and I2S interfaces are available if the second storage port (S1) is used in 8-bit mode. In this case, the configuration chosen should CY_U3P_IO_MATRIX_LPP_NONE.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PSPortMode_t

5.9.2.3 typedef enum CyU3PPartNumber_t CyU3PPartNumber_t

Enumeration of EZ-USB FX3 part numbers.

Description

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

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

See Also

CyU3PDeviceGetPartNumber

5.9.2.4 typedef enum CyU3PSPortMode_t CyU3PSPortMode_t

Define the various operating modes for the storage ports on the FX3S device.

Description

The FX3S device supports two storage ports which can be connected to SD/eMMC/SDIO peripherals. Each of these ports can be configured in a variety of modes. This enumeration lists the possible IO configurations for each of these storage ports.

The selected storage port IO configuration has some implications on the selection of other IO configurations like lppMode. Please see the FX3S device datasheet to identify the supported combinations.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PIoMatrixLppMode_t

5.9.3 Enumeration Type Documentation

5.9.3.1 enum CyU3PIoMatrixLppMode_t

Defines the enumerations for LPP IO line configurations.

Description

Most of the IO pins on the FX3 device are multi-purpose with the specific configuration being selected at system start-up. This enumeration lists the various IO operating modes relating to Serial peripheral interfaces.

The I2C interface is always available. If the GPIF data bus is configured as 32-bit wide, only one of the SPI, UART and I2S interfaces are available. In this case, the configuration chosen should be CY_U3P_IO_MATRIX_LPP_DEFAULT.

If the GPIF data bus is 8, 16 or 24 bits wide; it is possible to use all of the SPI, UART and I2S interfaces. However, the peripheral pins can be relocated in this case. Refer to the FX3 datasheet for more details and choose the desired configuration accordingly.

In the case of the FX3S device, none of the UART, SPI and I2S interfaces are available if the second storage port (S1) is used in 8-bit mode. In this case, the configuration chosen should CY_U3P_IO_MATRIX_LPP_NONE.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PSPortMode_t

Enumerator

CY_U3P_IO_MATRIX_LPP_DEFAULT Default LPP mode where all peripherals are enabled.

CY_U3P_IO_MATRIX_LPP_UART_ONLY LPP layout with GPIF 16-bit and UART only.

CY_U3P_IO_MATRIX_LPP_SPI_ONLY LPP layout with GPIF 16-bit and SPI only.

CY_U3P_IO_MATRIX_LPP_I2S_ONLY LPP layout with GPIF 16-bit and I2S only.

CY_U3P_IO_MATRIX_LPP_NONE FX3S specific configuration where UART, SPI and I2S are disabled.

5.9.3.2 enum CyU3PPartNumber_t

Enumeration of EZ-USB FX3 part numbers.

Description

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

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

See Also

CyU3PDeviceGetPartNumber

Enumerator

CYPART_USB3014 CYUSB3014: 512 KB RAM; GPIF can be 32 bit; OTG and USB host supported

CYPART_USB3012 CYUSB3012: 256 KB RAM; GPIF can be 32 bit

CYPART_USB3013 CYUSB3013: 512 KB RAM; GPIF - 16 bit bus only

CYPART_USB3011 CYUSB3011: 256 KB RAM; GPIF - 16 bit bus only

CYPART_USB3035 CYUSB3035: 512 KB RAM; GPIF - 16 bit bus; Supports two SD/eMMC/SDIO ports

CYPART_USB3034 CYUSB3034: 512 KB RAM; GPIF - 16 bit bus; Supports two SD/eMMC/SDIO ports

CYPART_USB3033 CYUSB3033: 512 KB RAM; GPIF - 16 bit bus; Supports single SD/eMMC/SDIO port

CYPART_USB3032 CYUSB3032: 256 KB RAM; GPIF - 16 bit bus; No USB host/OTG; Two storage ports

CYPART_USB3031 CYUSB3031: 256 KB RAM; GPIF - 16 bit bus; No USB host/OTG; Single storage port

CYPART_USB2035 CYUSB2035: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports

CYPART_USB2034 CYUSB2034: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports two SD/eMMC/SDIO ports

CYPART_USB2033 CYUSB2033: 512 KB RAM; USB 2.0 only; GPIF - 16 bit; Supports single SD/eMMC/SDIO port

CYPART_USB2032 CYUSB2032: 256 KB RAM; USB 2.0 only; GPIF - 16 bit; No USB host/OTG; Two storage ports

CYPART_USB2031 CYUSB2031: 256 KB RAM; USB 2.0 only; GPIF - 16 bit; No USB host/OTG; Single storage port

CYPART_USB3025 CYUSB3025: 512 KB RAM; No GPIF port; USB host/OTG; Two storage ports

CYPART_USB3024 CYUSB3024: 512 KB RAM; No GPIF port; USB host/OTG; Two storage ports

CYPART_USB3023 CYUSB3023: 512 KB RAM; No GPIF port; No USB host/OTG; Single storage port

CYPART_USB3021 CYUSB3021: 256 KB RAM; No GPIF port; No USB host/OTG; Single storage port

CYPART_USB2025 CYUSB2025: 512 KB RAM; USB 2.0 only; No GPIF port; USB host/OTG; Two storage ports

CYPART_USB2024 CYUSB2024: 512 KB RAM; USB 2.0 only; No GPIF port; USB host/OTG; Two storage ports

CYPART_USB2023 CYUSB2023: 512 KB RAM; USB 2.0 only; No GPIF port; No USB host/OTG; Single storage port

CYPART_USB3061 CYUSB3061: 256 KB RAM; Fixed Function GPIF; 1 MIPI-CSI lane

CYPART_USB3062 CYUSB3062: 512 KB RAM; Fixed Function GPIF; 1 MIPI-CSI lane

CYPART_USB3063 CYUSB3063: 256 KB RAM; Fixed Function GPIF; 2 MIPI-CSI lanes

CYPART_USB3064 CYUSB3064: 512 KB RAM; Fixed Function GPIF; 2 MIPI-CSI lanes

CYPART_USB3065 CYUSB3065: 512 KB RAM; Fixed Function GPIF; 4 MIPI-CSI lane

CYPART_USB2064 CYUSB2064: 512 KB RAM; USB 2.0 only; Fixed Function GPIF; 2 MIPI-CSI lane

CYPART_LASTDEV Unknown device type

5.9.3.3 enum CyU3PSPortMode_t

Define the various operating modes for the storage ports on the FX3S device.

Description

The FX3S device supports two storage ports which can be connected to SD/eMMC/SDIO peripherals. Each of these ports can be configured in a variety of modes. This enumeration lists the possible IO configurations for each of these storage ports.

The selected storage port IO configuration has some implications on the selection of other IO configurations like lppMode. Please see the FX3S device datasheet to identify the supported combinations.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PIoMatrixLppMode_t

Enumerator

CY_U3P_SPORT_INACTIVE Storage port not in use. Pins are available for other interfaces.

CY_U3P_SPORT_4BIT SD/MMC interface with four bit data bus. Upper half of data bus is available for other interfaces.

CY_U3P_SPORT_8BIT SD/MMC interface with eight bit data bus. All pins (CLK, CMD, D7-D0) are part of storage interface.

5.9.4 Function Documentation

5.9.4.1 void CyFx3DevClearSwInterrupt (void)

Clear a FX3 device specific software interrupt.

Description

This function clears a device specific software interrupt. The software interrupt mechanism is not used in the SDK.

Return Value

None

5.9.4.2 uint8_t CyFx3DevGetMipiLaneCount (void)

Get the number of CSI lanes supported by the CX3 part in use.

Description

Different CX3 parts can support different number of CSI lanes through image sensors can be connected. This API returns the number of CSI lanes that are supported by the CX3 part in use..

Return Value

Number of CSI lanes supported.

5.9.4.3 CyU3PPartNumber_t CyFx3DevIdentifyPart (void)

Get the current FX3/FX3S device part number.

Description

This function gets the current FX3/FX3S part number by querying the device identification registers. This function can be used by the code to avoid calling functions that would cause failures because the Silicon does not support them.

Return Value

Device part number

5.9.4.4 void CyFx3DevInitPageTables (void)

Setup the default page tables for the FX3 device.

Description

The MMU on the FX3/FX3S device needs to be turned on for the data cache to work. This function sets up a default set of page tables that map each valid physical address to the corresponding virtual address, so that the device can operate with the MMU turned on.

Return Value

None

5.9.4.5 CyBool_t CyFx3DevIOConfigure (CyU3PIoMatrixConfig_t * *cfg_p*)

Configure the IO matrix on the device.

Description

The FX3 and other devices in the family have a configurable IO matrix which allows multiplexing of multiple functions onto a IO pin at different times. This function validates the IO configuration specified by the user, and applies it if valid.

Return Value

CyTrue if the IO configuration is valid for the device and has been applied. CyFalse if the IO configuration is illegal or inconsistent.

Parameters

cfg_p	IO configuration parameters.

5.9.4.6 CyBool_t CyFx3DevIOIsGpio (uint32_t *gpioId,* CyBool_t *isSimple*)

Check if the specified GPIO is configured as a GPIO.

Description

This function checks whether the specified FX3/FX3S GPIO pin has been configured as a GPIO of the specified type (simple or complex).

Return Value

CyTrue if the pin is currently configured as a GPIO of the specified type. CyFalse if the pin is invalid or if it is not configured as a GPIO.

Parameters

gpioId	GPIO whose state is to be queried.
isSimple	Whether to check for a simple GPIO (CyTrue) or a complex GPIO (CyFalse).

5.9.4.7 CyBool_t CyFx3DevIOIsI2cConfigured (void)

Check if I2C is enabled in the current IO configuration.

Description

This function checks whether the I2C interface is enabled in the current IO configuration of the FX3 device.

Return Value

CyTrue if I2C is enabled. CyFalse if I2C is disabled.

5.9.4.8 CyBool_t CyFx3DevIOIsI2sConfigured (void)

Check if I2S is enabled in the current IO configuration.

Description

This function checks whether the I2S interface is enabled in the current IO configuration of the FX3 device.

Return Value

CyTrue if I2S is enabled. CyFalse if I2S is disabled.

5.9.4.9 CyBool_t CyFx3DevIOIsSib0Configured (void)

Check if the SIB0 port is configured.

Description

Check whether the SIB0 port is enabled in the current IO configuration of the FX3S/SD3 device.

Return Value

CyTrue if the SIB0 port is enabled in the IO configuration. CyFalse if the SIB0 port is not enabled in the IO configuration.

5.9.4.10 CyBool_t CyFx3DevIOIsSib1Configured (void)

Check if the SIB1 port is configured.

Description

Check whether the SIB1 port is enabled in the current IO configuration of the FX3S/SD3 device.

Return Value

CyTrue if the SIB1 port is enabled in the IO configuration. CyFalse if the SIB1 port is not enabled in the IO configuration.

5.9.4.11 CyBool_t CyFx3DevIOIsSib8BitWide (uint8_t *portId*)

Check if the specified SIB port is configured for 8 bit operation.

Description

The SIB ports on the FX3S/SD3 device can be configured for 4 bit or 8 bit data operation. This function checks whether the current IO configuration of the specified storage port allows 8 bit operation.

Return Value

CyTrue if 8 bit operation is allowed. CyFalse if 8 bit operation is not allowed.

Parameters

portId	Port being queried (0 or 1).

5.9.4.12 CyBool_t CyFx3DevIOIsSpiConfigured (void)

Check if SPI is enabled in the current IO configuration.

Description

This function checks whether the SPI interface is enabled in the current IO configuration of the FX3 device.

Return Value

CyTrue if SPI is enabled. CyFalse if SPI is disabled.

5.9.4.13 CyBool_t CyFx3DevIOIsUartConfigured (void)

Check if UART is enabled in the current IO configuration.

Description

This function checks whether the UART interface is enabled in the current IO configuration of the FX3 device.

Return Value

CyTrue if UART is enabled. CyFalse if UART is disabled.

5.9.4.14 void CyFx3DevIOSelectGpio (uint8_t *gpioId,* CyBool_t *enable,* CyBool_t *isSimple*)

Control the override of a IO pin as a simple or complex GPIO.

Description

IO pins on the FX3/FX3S interface can be temporarily over-ridden as simple or complex GPIOs during device operation. This API is used to place or remove these GPIO override modes.

Return Value

None

Parameters

gpioId	GPIO to be modified.
enable	Whether the override is to be enabled (CyTrue) or removed (CyFalse).
isSimple	Whether over-riding as a simple (CyTrue) or a complex (CyFalse) GPIO. Is don't care when enable is CyFalse.

5.9.4.15 CyBool_t CyFx3DevIsGpif32Supported (void)

Check if the part supports a 32 bit wide GPIF II data bus.

Description

Some FX3 parts as well as the FX3S devices can only make use of the GPIF II interface with a 16 bit or narrower data bus. This function checks whether the part in use can support a 24 bit or 32 bit wide GPIF II data bus.

Return Value

CyTrue if GPIF II data bus can be 32 bits wide. CyFalse if GPIF II data bus can only be 16 bits or smaller.

5.9.4.16 CyBool_t CyFx3DevIsGpifConfigurable (void)

Check if the part supports a configurable GPIF interface.

Description

The CX3 parts make use of a fixed function GPIF mode, and do not support dynamic GPIF configuration. This API is used to check whether the part in use supports GPIF configuration.

Return Value

CyTrue if the GPIF interface is configurable. CyFalse if the GPIF interface is not configurable.

5.9.4.17 CyBool_t CyFx3DevIsGpifSupported (void)

Check if the part supports the PIB/GPIF II interface.

Description

The FX3 SDK supports parts such as the SD3 and SD2 devices which do not provide a GPIF II interface. This function is used to check whether the part in use can support GPIF II functionality.

Return Value

CyTrue if GPIF II (PIB) is supported. CyFalse if GPIF II is not supported.

5.9.4.18 CyBool_t CyFx3DevIsI2sSupported (void)

Check if the part supports the I2S interface.

Description

Some parts in the FX3 family do not support the I2S interface. This function checks whether I2S is supported by the part in use.

Return Value

CyTrue if I2S interface is supported. CyFalse if I2S interface is not supported.

5.9.4.19 CyBool_t CyFx3DevIsMipicsiSupported (void)

Check if the part supports the MIPI CSI interface.

Description

Some parts in the FX3 family (CX3) support a MIPI CSI interface through which an image sensor can be connected. This function checks whether the part in use supports the CSI interface.

Return Value

CyTrue if the CSI interface is supported. CyFalse if the CSI interface is not supported.

5.9.4.20 CyBool_t CyFx3DevIsOtgSupported (void)

Check if the part being used supports the USB OTG and USB 2.0 host functionality.

Description

Some devices in the FX3/FX3S product family do not support the USB 2.0 host and OTG functionality. This function is used to check whether the part being used supports the host/OTG functionality.

Return Value

CyTrue if USB 2.0 host and OTG is supported. CyFalse if USB 2.0 host and OTG is not supported.

5.9.4.21 CyBool_t CyFx3DevIsRam512Supported (void)

Check if the part being used supports 512 KB of System RAM.

Description

Some devices in the FX3/FX3S product family support only 256 KB of System RAM, and the firmware needs to be constrained to work with the available memory. This function is used to check whether the part being used supports 512 KB of System RAM.

Return Value

CyTrue if the part supports 512 KB of RAM. CyFalse if the part supports only 256 KB of RAM.

5.9.4.22 CyBool_t CyFx3DevIsSib0Supported (void)

Check if the part supports the S0 storage port.

Description

Some parts in the FX3 family (FX3S, SD3 and SD2) support a storage port through which SD cards, eMMC devices or SDIO cards can be connected. This function checks whether the part in use supports the S0 storage port.

Return Value

CyTrue if the S0 storage port is supported. CyFalse if the S0 storage port is not supported.

5.9.4.23 CyBool_t CyFx3DevIsSib1Supported (void)

Check if the part supports the S1 storage port.

Description

Some parts in the FX3 family (FX3S, SD3 and SD2) support two storage ports through which SD cards, eMMC devices or SDIO cards can be connected. This function checks whether the part in use supports the S1 (second) storage port.

Return Value

CyTrue if the S1 storage port is supported. CyFalse if the S1 storage port is not supported.

5.9.4.24 CyBool_t CyFx3DevIsUsb3Supported (void)

Check if the part being used supports the USB 3.0 peripheral functionality.

Description

Some low cost devices in the FX3 family (such as SD2 or FX2G2) do not support the USB 3.0 peripheral functionality, while retaining all of the other capabilities such as the ARM9 core, the high-performance distributed DMA architecture, the GPIF II port and the peripheral support. This function checks whether the active part supports the USB 3.0 function.

Return Value

CyTrue if USB 3.0 is supported. CyFalse if USB 3.0 is not supported.

5.9.4.25 void CyFx3PibDllEnable (void)

Enable the DLL in the PIB block.

Description

The PIB block on FX3 has an embedded DLL that can be used when FX3 is acting as a slave device with an incoming clock signal. This function enables the DLL in the PIB block.

Return Value

None

5.9.4.26 uint16_t CyFx3PibGetDllStatus (void)

Get the current status of the PIB DLL.

Description

Get the locked/unlocked status of the PIB DLL.

Return Value

None

5.9.4.27 void CyFx3PibPowerOff (void)

Power off the PIB and GPIF II blocks on the FX3/FX3S device.

Description

A device specific sequence needs to be followed when powering the PIB and GPIF II blocks on the FX3 device off. This function implements the sequence.

Note: The PIB clock should only be disabled after this function returns.

Return value

None

5.9.4.28 void CyFx3PibPowerOn (void)

Power on the PIB and GPIF II blocks on the FX3/FX3S device.

Description

A device specific sequence needs to be followed when powering the PIB and GPIF II blocks on the FX3 device on. This function implements the sequence.

Note: The PIB clock should have been enabled before calling this function.

Return value

None

5.9.4.29 void CyFx3SibPowerOff (void)

Power the storage interface block on the FX3S/SD3 device off.

Description

This function powers the storage interface block (SIB) on the FX3S/SD3 device off. The SIB clock(s) can be turned off after calling this function.

Return Value

None

5.9.4.30 void CyFx3SibPowerOn (void)

Power the storage interface block on the FX3S/SD3 device on.

Description

This function powers the storage interface block (SIB) on the FX3S/SD3 device on. A common power-up sequence is used for all available storage ports. Clocks for the available storage ports need to be enabled before calling this function.

Return Value

None

5.9.4.31 void CyFx3Usb2PhySetup (void)

Configure the USB 2.0 PHY on FX3.

Description

This function sets up USB 2.0 PHY on the FX3 for operation. The settings made by this API are required for passing USB 2.0 electrical tests.

Return Value

None

5.9.4.32 void CyFx3Usb3LnkRelaxHpTimeout (void)

Relax the USB 3.0 HP ACK Timeout period.

Description

In some cases, FX3 does not receive the ACK for a USB 3.0 link header in time, and gets into the SS.Inactive state. This API relaxes the timeout for the ACK so that this kind of error is prevented.

Return Value

None

5.9.4.33 void CyFx3Usb3LnkSetup (void)

Configure the USB 3.0 LINK on FX3.

Description

This function sets up the USB 3.0 LINK on the FX3 for operation. The settings made by this API are required for passing USB 3.0 link layer tests.

Return Value

None

5.9.4.34 void CyFx3Usb3SendTP (uint32_t * *tpData*)

Send a USB 3.0 Transaction Packet.

Description

This function sends a USB 3.0 Transaction Packet (TP) with user specified data to the USB host.

Return Value

None

Parameters

tpData	Pointer to the TP data (3 dwords) to be sent.

5.9.4.35 void CyFx3UsbDmaPrefetchEnable (CyBool_t *streamEnable*)

Enable data prefetch from the DMA sockets on the USB egress (IN) data path.

Description

This function enables prefetching of data from the DMA sockets on the USB egress (or IN endpoint) data path, so as to obtain better transfer performance. This setting is always enabled during USB 2.0 operation, but should be used with caution on a USB 3.0 connection.

Return Value

None

Parameters

streamEnable	Enable stream mode of data prefetch. Can cause endpoint freeze if used with endpoints that send a lot of short packets.

5.9.4.36 void CyFx3UsbPowerOn (void)

Power on the USB block on the FX3/FX3S device.

Description

A device specific sequence needs to be followed when powering the USB block on the FX3 device on. This function implements the sequence.

Return value

None

5.9.4.37 void CyFx3UsbWritePhyReg (uint16_t *phyAddr,* uint16_t *phyVal*)

Write to a USB 3.0 PHY Control Register.

Description

The USB 3.0 PHY on the FX3 device has a few control registers that may need to be updated at various stages of device operation. This function writes a user specified value into one of the PHY control registers.

Return Value

None

Parameters

phyAddr	Address of the PHY register.
phyVal	Value to be written.

5.10 firmware/u3p_firmware/inc/cyfxapidesc.h File Reference

Provides a brief description of the FX3 APIs.

5.10.1 Detailed Description

Provides a brief description of the FX3 APIs.

5.11 firmware/u3p_firmware/inc/cyfxversion.h File Reference

Version information for the FX3 API library.

Macros

- #define CYFX_VERSION_MAJOR (1)

  Major number of the release version.
- #define CYFX_VERSION_MINOR (3)

  Minor number of the release version.
- #define CYFX_VERSION_PATCH (1)

  Patch number for this release.
- #define CYFX_VERSION_BUILD (122)

  Build number.

5.11.1 Detailed Description

Version information for the FX3 API library. **Description**

The version information is composed of four values.

Note

The version information is compiled into the library and is provided here for reference.

These values should not be changed and can only be used to conditionally enable code.

5.12 firmware/u3p_firmware/inc/cyu3cardmgr_fx3s.h File Reference

This file contains the provides the low level SD/MMC/SDIO driver related macros and definitions.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3sib.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Macros

- #define CY_U3P_SD_MMC_CMD0_GO_IDLE_STATE (0x00)

  The following macros define the various SD and MMC commands that are defined in the SD and MMC device specifications.
- #define **CY_U3P_MMC_CMD1_SEND_OP_COND** (0x01)
- #define **CY_U3P_SD_MMC_CMD2_ALL_SEND_CID** (0x02)
- #define **CY_U3P_SD_CMD3_SEND_RELATIVE_ADDR** (0x03)
- #define **CY_U3P_MMC_CMD3_SET_RELATIVE_ADDR** (0x03)
- #define **CY_U3P_SD_MMC_CMD4_SET_DSR** (0x04)
- #define **CY_U3P_MMC_CMD5_SLEEP_AWAKE** (0x05)
- #define **CY_U3P_SD_MMC_CMD5_IO_SEND_OP_COND** (0x05)
- #define **CY_U3P_SD_MMC_CMD6_SWITCH** (0x06)
- #define **CY_U3P_SD_MMC_CMD7_SELECT_DESELECT_CARD** (0x07)
- #define **CY_U3P_SD_CMD8_SEND_IF_COND** (0x08)
- #define **CY_U3P_MMC_CMD8_SEND_EXT_CSD** (0x08)
- #define **CY_U3P_SD_MMC_CMD9_SEND_CSD** (0x09)
- #define **CY_U3P_SD_MMC_CMD10_SEND_CID** (0x0A)
- #define **CY_U3P_MMC_CMD11_READ_DAT_UNTIL_STOP** (0x0B)
- #define **CY_U3P_SD_CMD11_VOLTAGE_SWITCH** (0x0B)
- #define **CY_U3P_SD_MMC_CMD12_STOP_TRANSMISSION** (0x0C)
- #define **CY_U3P_SD_MMC_CMD13_SEND_STATUS** (0x0D)
- #define **CY_U3P_MMC_CMD14_BUSTEST_R** (0x0E)
- #define **CY_U3P_SD_MMC_CMD15_GO_INACTIVE_STATE** (0x0F)
- #define **CY_U3P_SD_MMC_CMD16_SET_BLOCKLEN** (0x10)
- #define **CY_U3P_SD_MMC_CMD17_READ_SINGLE_BLOCK** (0x11)
- #define **CY_U3P_SD_MMC_CMD18_READ_MULTIPLE_BLOCK** (0x12)
- #define **CY_U3P_MMC_CMD19_BUSTEST_W** (0x13)
- #define **CY_U3P_SD_CMD19_SEND_TUNING_PATTERN** (0x13)
- #define **CY_U3P_MMC_CMD20_WRITE_DAT_UNTIL_STOP** (0x14)
- #define **CY_U3P_SD_MMC_CMD21_RESERVED_21** (0x15)
- #define **CY_U3P_SD_MMC_CMD22_RESERVED_22** (0x16)
- #define **CY_U3P_MMC_CMD23_SET_BLOCK_COUNT** (0x17)
- #define **CY_U3P_SD_MMC_CMD24_WRITE_BLOCK** (0x18)
- #define **CY_U3P_SD_MMC_CMD25_WRITE_MULTIPLE_BLOCK** (0x19)
- #define **CY_U3P_MMC_CMD26_PROGRAM_CID** (0x1A)
- #define **CY_U3P_SD_MMC_CMD27_PROGRAM_CSD** (0x1B)
- #define **CY_U3P_SD_MMC_CMD28_SET_WRITE_PROT** (0x1C)
- #define **CY_U3P_SD_MMC_CMD29_CLR_WRITE_PROT** (0x1D)
- #define **CY_U3P_SD_MMC_CMD30_SEND_WRITE_PROT** (0x1E)
- #define **CY_U3P_SD_MMC_CMD31_RESERVED** (0x1F)
- #define **CY_U3P_SD_CMD32_ERASE_WR_BLK_START** (0x20)
- #define **CY_U3P_SD_CMD33_ERASE_WR_BLK_END** (0x21)
- #define **CY_U3P_SD_MMC_CMD34_RESERVED** (0x22)
- #define **CY_U3P_MMC_CMD35_ERASE_GROUP_START** (0x23)
- #define **CY_U3P_MMC_CMD36_ERASE_GROUP_END** (0x24)
- #define **CY_U3P_SD_MMC_CMD37_RESERVED** (0x25)
- #define **CY_U3P_SD_MMC_CMD38_ERASE** (0x26)
- #define **CY_U3P_MMC_CMD39_FAST_IO** (0x27)
- #define **CY_U3P_MMC_CMD40_GO_IRQ_STATE** (0x28)
- #define **CY_U3P_SD_MMC_CMD41_RESERVED** (0x29)
- #define **CY_U3P_SD_MMC_CMD42_LOCK_UNLOCK** (0x2A)
- #define **CY_U3P_SD_MMC_CMD43_RESERVED** (0x2B)
- #define **CY_U3P_SD_MMC_CMD44_RESERVED** (0x2C)
- #define **CY_U3P_SD_MMC_CMD45_RESERVED** (0x2D)

- #define **CY_U3P_SD_MMC_CMD46_RESERVED** (0x2E)
- #define **CY_U3P_SD_MMC_CMD47_RESERVED** (0x2F)
- #define **CY_U3P_SD_MMC_CMD48_RESERVED** (0x30)
- #define **CY_U3P_SD_MMC_CMD49_RESERVED** (0x31)
- #define **CY_U3P_SD_MMC_CMD50_RESERVED** (0x32)
- #define **CY_U3P_SD_MMC_CMD51_RESERVED** (0x33)
- #define **CY_U3P_SD_MMC_CMD52_IO_RW_DIRECT** (0x34)
- #define **CY_U3P_SD_MMC_CMD53_IO_RW_EXTENDED** (0x35)
- #define **CY_U3P_SD_MMC_CMD54_RESERVED** (0x36)
- #define **CY_U3P_SD_MMC_CMD55_APP_CMD** (0x37)
- #define **CY_U3P_SD_MMC_CMD56_GEN_CMD** (0x38)
- #define **CY_U3P_SD_MMC_CMD57_RESERVED** (0x39)
- #define **CY_U3P_SD_MMC_CMD58_RESERVED** (0x3A)
- #define **CY_U3P_SD_MMC_CMD59_RESERVED** (0x3B)
- #define **CY_U3P_SD_MMC_CMD60_RW_MULTIPLE_REGISTER** (0x3C)
- #define **CY_U3P_SD_MMC_CMD61_RW_MULTIPLE_BLOCK** (0x3D)
- #define **CY_U3P_SD_ACMD6_SET_BUS_WIDTH** (0x06)
- #define **CY_U3P_SD_ACMD13_SD_STATUS** (0x0D)
- #define **CY_U3P_SD_ACMD17_RESERVED** (0x11)
- #define **CY_U3P_SD_ACMD18_RESERVED** (0x12)
- #define **CY_U3P_SD_ACMD19_RESERVED** (0x13)
- #define **CY_U3P_SD_ACMD20_RESERVED** (0x14)
- #define **CY_U3P_SD_ACMD21_RESERVED** (0x15)
- #define **CY_U3P_SD_ACMD22_SEND_NUM_WR_BLOCKS** (0x16)
- #define **CY_U3P_SD_ACMD23_SET_WR_BLK_ERASE_COUNT** (0x17)
- #define **CY_U3P_SD_ACMD24_RESERVED** (0x18)
- #define **CY_U3P_SD_ACMD25_RESERVED** (0x19)
- #define **CY_U3P_SD_ACMD26_RESERVED** (0x1A)
- #define **CY_U3P_SD_ACMD38_RESERVED** (0x26)
- #define **CY_U3P_SD_ACMD39_RESERVED** (0x27)
- #define **CY_U3P_SD_ACMD40_RESERVED** (0x28)
- #define **CY_U3P_SD_ACMD41_SD_SEND_OP_COND** (0x29)
- #define **CY_U3P_SD_ACMD42_SET_CLR_CARD_DETECT** (0x2A)
- #define **CY_U3P_SD_ACMD43_RESERVED** (0x2B)
- #define **CY_U3P_SD_ACMD49_RESERVED** (0x31)
- #define **CY_U3P_SD_ACMD51_SEND_SCR** (0x33)
- #define CY_U3P_SD_MMC_R1_RESP_BITS (0x26)

  These macros define constants denoting the length in bits of various response types defined in the SD and MMC specifications. These lengths are excluding the Start, direction and CRC bits.
- #define **CY_U3P_SD_MMC_R1B_RESP_BITS** (0x26)
- #define **CY_U3P_SD_MMC_R2_RESP_BITS** (0x7E)
- #define **CY_U3P_SD_MMC_R3_RESP_BITS** (0x26)
- #define **CY_U3P_SD_MMC_R4_RESP_BITS** (0x26)
- #define **CY_U3P_SD_MMC_R5_RESP_BITS** (0x26)
- #define **CY_U3P_SD_R6_RESP_BITS** (0x26)
- #define **CY_U3P_SD_R7_RESP_BITS** (0x26)
- #define **CY_U3P_SD_MMC_REQD_CMD_CLASS** (0x0005) /∗ Classes 0, 2 are mandatory. ∗/
- #define **CY_U3P_SD_MMC_WRITE_CMD_CLASS** (0x0010) /∗ Class 4. ∗/
- #define **CY_U3P_SD_MMC_ERASE_CMD_CLASS** (0x0020) /∗ Class 5. ∗/
- #define CY_U3P_SD_REG_OCR_LEN (4)

  Length of the OCR register in bytes.
- #define CY_U3P_SD_REG_CID_LEN (16)

  Length of the CID register in bytes.
- #define CY_U3P_SD_REG_CSD_LEN (16)

Length of the CSD register in bytes.

- #define CY_U3P_SDIO_SUSPEND (0x0)

  SDIO operation code for function suspend.
- #define CY_U3P_SDIO_RESUME (0x1)

  SDIO operation code for function resume.
- #define CY_U3P_SDIO_READ (0x0)

  Operation code for read from SDIO card.
- #define CY_U3P_SDIO_WRITE (0x1)

  Operation code for write to SDIO card.
- #define CY_U3P_SDIO_RW_BYTE_MODE (0x0)

  Flag that indicates that the SDIO transfer requested is a byte-mode request.
- #define CY_U3P_SDIO_RW_BLOCK_MODE (0x1)

  Flag that indicates that the SDIO transfer request is a block-mode request.
- #define CY_U3P_SDIO_RW_ADDR_FIXED (0x0)

  Flag that indicates that the SDIO transfer should be performed with a fixed address.
- #define CY_U3P_SDIO_RW_ADDR_AUTO_INCREMENT (0x1)

  Flag that indicates that the address for the SDIO transfer should be auto incremented.
- #define CY_U3P_SDIO_REG_CCCR_REVISION (0x00)

  Address for various bytes in the CCCR.
- #define CY_U3P_SDIO_REG_SD_SPEC_REVISION (0x01)
- #define CY_U3P_SDIO_REG_IO_ENABLE (0x02)
- #define CY_U3P_SDIO_REG_IO_READY (0x03)
- #define CY_U3P_SDIO_REG_IO_INTR_ENABLE (0x04)
- #define CY_U3P_SDIO_REG_IO_INTR_PENDING (0x05)
- #define CY_U3P_SDIO_REG_IO_ABORT (0x06)
- #define CY_U3P_SDIO_REG_BUS_INTERFACE_CONTROL (0x07)
- #define CY_U3P_SDIO_REG_CARD_CAPABILITY (0x08)
- #define CY_U3P_SDIO_REG_CIS_PTR_D0 (0x09)
- #define CY_U3P_SDIO_REG_CIS_PTR_D1 (0x0A)
- #define CY_U3P_SDIO_REG_CIS_PTR_D2 (0x0B)
- #define CY_U3P_SDIO_REG_BUS_SUSPEND (0x0C)
- #define CY_U3P_SDIO_REG_FUNCTION_SELECT (0x0D)
- #define CY_U3P_SDIO_REG_EXEC_FLAGS (0x0E)
- #define CY_U3P_SDIO_REG_READY_FLAGS (0x0F)
- #define CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D0 (0x10)
- #define CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D1 (0x11)
- #define CY_U3P_SDIO_REG_POWER_CONTROL (0x12)
- #define CY_U3P_SDIO_REG_CCCR_HIGH_SPEED (0x13)
- #define CY_U3P_SDIO_REG_UHS_I_SUPPORT (0x14)
- #define CY_U3P_SDIO_REG_DRIVER_STRENGTH (0x15)
- #define CY_U3P_SDIO_REG_INTERRUPT_EXTENSION (0x16)
- #define CY_U3P_SDIO_REG_FBR_INTERFACE_CODE (0x00)

  Address for various bytes in the FBR.
- #define CY_U3P_SDIO_REG_FBR_EXT_INTERFACE_CODE (0x01)
- #define CY_U3P_SDIO_REG_FBR_POWER_SELECT (0x02)
- #define CY_U3P_SDIO_REG_FBR_CIS_PTR_DO CY_U3P_SDIO_REG_CIS_PTR_D0
- #define CY_U3P_SDIO_REG_FBR_CIS_PTR_D1 CY_U3P_SDIO_REG_CIS_PTR_D1
- #define CY_U3P_SDIO_REG_FBR_CIS_PTR_D2 CY_U3P_SDIO_REG_CIS_PTR_D2
- #define CY_U3P_SDIO_REG_FBR_CSA_PTR_DO (0x0C)
- #define CY_U3P_SDIO_REG_FBR_CSA_PTR_D1 (0x0D)
- #define CY_U3P_SDIO_REG_FBR_CSA_PTR_D2 (0x0E)
- #define CY_U3P_SDIO_REG_FBR_DATA_ACCESS_WINDOW (0x0F)

- #define CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D0 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D0
- #define CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D1 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D1
- #define CY_U3P_SDIO_LOW_SPEED (0x00)
- #define CY_U3P_SDIO_FULL_SPEED (0x01)
- #define CY_U3P_SDIO_HIGH_SPEED (0x02)
- #define CY_U3P_SDIO_SSDR12_SPEED (0x04)
- #define CY_U3P_SDIO_SSDR25_SPEED (0x10)
- #define CY_U3P_SDIO_SSDR50_SPEED (0x20)
- #define CY_U3P_SDIO_SSDR104_SPEED (0x40)
- #define CY_U3P_SDIO_SDDR50_SPEED (0x80)
- #define CY_U3P_SDIO_CARD_CAPABLITY_SDC (0x01)

  Bit fields for various SDIO card capability codes.
- #define CY_U3P_SDIO_CARD_CAPABLITY_SMB (0x02)
- #define CY_U3P_SDIO_CARD_CAPABLITY_SRW (0x04)
- #define CY_U3P_SDIO_CARD_CAPABLITY_SBS (0x08)
- #define CY_U3P_SDIO_CARD_CAPABLITY_S4MI (0x10)
- #define CY_U3P_SDIO_CARD_CAPABLITY_E4MI (0x20)
- #define CY_U3P_SDIO_CARD_CAPABLITY_LSC (0x40)
- #define CY_U3P_SDIO_CARD_CAPABLITY_4BLS (0x80)
- #define CY_U3P_SDIO_READ_AFTER_WRITE (0x01)
- #define CY_U3P_SDIO_SUPPORT_HIGH_SPEED (0x01)
- #define CY_U3P_SDIO_ENABLE_HIGH_SPEED (0x02)
- #define CY_U3P_SDIO_RESET (0x08)
- #define **CY_U3P_SDIO_CIA_FUNCTION** (0x00)
- #define CY_U3P_SDIO_CISTPL_NULL (0x00)

  List of major SDIO Tuple Codes.
- #define CY_U3P_SDIO_CISTPL_END (0xFF)
- #define CY_U3P_SDIO_CISTPL_MANFID (0x20)
- #define CY_U3P_SDIO_CISTPL_FUNCE (0x22)
- #define CY_U3P_SDIO_CCCR_Version_1_00 (0x00)

  CCCR Format Version Numbers.
- #define CY_U3P_SDIO_CCCR_Version_1_10 (0x01)
- #define CY_U3P_SDIO_CCCR_Version_2_00 (0x02)
- #define CY_U3P_SDIO_CCCR_Version_3_00 (0x03)
- #define CY_U3P_SDIO_Version_1_00 (0x00)

  SDIO Specification Revision Numbers.
- #define CY_U3P_SDIO_Version_1_10 (0x01)
- #define CY_U3P_SDIO_Version_1_20 (0x02)
- #define CY_U3P_SDIO_Version_2_00 (0x03)
- #define CY_U3P_SDIO_Version_3_00 (0x04)
- #define CY_U3P_SDIO_SD_Version_1_00 (0x00)

  SDIO SD Physical Layer Specifications.
- #define CY_U3P_SDIO_SD_Version_1_10 (0x10)
- #define CY_U3P_SDIO_SD_Version_2_00 (0x20)
- #define CY_U3P_SDIO_SD_Version_3_00 (0x30)
- #define CY_U3P_SDIO_UHS_SSDR50 (0x01)

  SDIO UHS-I support flags.
- #define CY_U3P_SDIO_UHS_SSDR104 (0x02)
- #define CY_U3P_SDIO_UHS_SDDR50 (0x04)
- #define CY_U3P_SDIO_SAI (0x01)

  SDIO Interrupt Extension flags.
- #define CY_U3P_SDIO_EAI (0x02)

- #define CY_U3P_SDIO_INTFC_NONE (0x00)

  SDIO Standard Function Interface Codes (as defined by SDIO 3.0 Spec).
- #define CY_U3P_SDIO_INTFC_UART (0x01)
- #define CY_U3P_SDIO_INTFC_BT_A (0x02)
- #define CY_U3P_SDIO_INTFC_BT_B (0x03)
- #define CY_U3P_SDIO_INTFC_GPS (0x04)
- #define CY_U3P_SDIO_INTFC_CAM (0x05)
- #define CY_U3P_SDIO_INTFC_PHS (0x06)
- #define CY_U3P_SDIO_INTFC_WLAN (0x07)
- #define CY_U3P_SDIO_INTFC_EMBD_ATA (0x08)
- #define CY_U3P_SDIO_INTFC_BT_A_AMP (0x09)
- #define CY_U3P_SDIO_INT_MASTER (0x00)

  SDIO Interrupt Enable and Disable Flags.
- #define CY_U3P_SDIO_DISABLE_INT (0x00)
- #define CY_U3P_SDIO_ENABLE_INT (0x01)
- #define **CY_U3P_SDIO_CHECK_INT_ENABLE_REG** (0x02)
- #define **CY_U3P_SDIO_ENABLE_ASYNC_INT** (0x03)
- #define CyU3PSdioInterruptBit(funcNo) ((0x01) << (funcNo))

  Macro to generate interrupt enable bit mask for a SDIO function.
- #define CyU3PSdioCheckInterruptEnableStatus(funcNo, ienReg) ((CyU3PSdioInterruptBit(funcNo) & (ienReg)) ? (1) : (0))

  Macro to check if interrupts are enabled or disabled in the returned data of the CyU3PSdioInterruptControl call.

5.12.1 Detailed Description

This file contains the provides the low level SD/MMC/SDIO driver related macros and definitions.

5.12.2 Macro Definition Documentation

5.12.2.1 #define CY_U3P_SD_MMC_CMD0_GO_IDLE_STATE (0x00)

The following macros define the various SD and MMC commands that are defined in the SD and MMC device specifications.

5.12.3 SD and MMC card commands

5.12.3.1 #define CY_U3P_SD_MMC_R1_RESP_BITS (0x26)

These macros define constants denoting the length in bits of various response types defined in the SD and MMC specifications. These lengths are excluding the Start, direction and CRC bits.

5.12.4 SD and MMC card response length

5.12.4.1 #define CY_U3P_SDIO_CARD_CAPABLITY_4BLS (0x80)

Low speed card with 4-bit support.

5.12.4.2 #define CY_U3P_SDIO_CARD_CAPABLITY_E4MI (0x20)

Enable Block Gap Interrupt.

5.12.4.3 #define CY_U3P_SDIO_CARD_CAPABLITY_LSC (0x40)

Low Speed Card.

5.12.4.4 #define CY_U3P_SDIO_CARD_CAPABLITY_S4MI (0x10)

Support Block Gap Interrupt.

5.12.4.5 #define CY_U3P_SDIO_CARD_CAPABLITY_SBS (0x08)

Support Bus Control.

5.12.4.6 #define CY_U3P_SDIO_CARD_CAPABLITY_SDC (0x01)

Bit fields for various SDIO card capability codes.
Support direct command.

5.12.4.7 #define CY_U3P_SDIO_CARD_CAPABLITY_SMB (0x02)

Support Multi-Block transfer.

5.12.4.8 #define CY_U3P_SDIO_CARD_CAPABLITY_SRW (0x04)

Support Read Wait.

5.12.4.9 #define CY_U3P_SDIO_CCCR_Version_1_00 (0x00)

CCCR Format Version Numbers.
CCCR version 1.00

5.12.4.10 #define CY_U3P_SDIO_CCCR_Version_1_10 (0x01)

CCCR version 1.10

5.12.4.11 #define CY_U3P_SDIO_CCCR_Version_2_00 (0x02)

CCCR version 2.00

5.12.4.12 #define CY_U3P_SDIO_CCCR_Version_3_00 (0x03)

CCCR version 3.00

5.12.4.13 #define CY_U3P_SDIO_CISTPL_END (0xFF)

Tuple Chain End

5.12.4.14 #define CY_U3P_SDIO_CISTPL_FUNCE (0x22)

Function Extensions.

5.12.4.15 #define CY_U3P_SDIO_CISTPL_MANFID (0x20)

Manufacturer ID

5.12.4.16 #define CY_U3P_SDIO_CISTPL_NULL (0x00)

List of major SDIO Tuple Codes.

Starting Tuple Code

5.12.4.17 #define CY_U3P_SDIO_DISABLE_INT (0x00)

Master interrupt disabled.

5.12.4.18 #define CY_U3P_SDIO_EAI (0x02)

Enable Asynchronous Interrupt.

5.12.4.19 #define CY_U3P_SDIO_ENABLE_HIGH_SPEED (0x02)

Enable high speed flag.

5.12.4.20 #define CY_U3P_SDIO_ENABLE_INT (0x01)

Master interrupt enabled.

5.12.4.21 #define CY_U3P_SDIO_FULL_SPEED (0x01)

Full speed.

5.12.4.22 #define CY_U3P_SDIO_HIGH_SPEED (0x02)

High speed.

5.12.4.23 #define CY_U3P_SDIO_INT_MASTER (0x00)

SDIO Interrupt Enable and Disable Flags.

Master interrupt flag.

5.12.4.24 #define CY_U3P_SDIO_INTFC_BT_A (0x02)

Function supports SDIO Bluetooth Type-A standard interface.

5.12.4.25 #define CY_U3P_SDIO_INTFC_BT_A_AMP (0x09)

Function supports SDIO Bluetooth Type-A AMP standard interface.

5.12.4.26 #define CY_U3P_SDIO_INTFC_BT_B (0x03)

Function supports SDIO Bluetooth Type-B standard interface.

5.12.4.27 #define CY_U3P_SDIO_INTFC_CAM (0x05)

Function supports SDIO Camera standard interface.

5.12.4.28 #define CY_U3P_SDIO_INTFC_EMBD_ATA (0x08)

Function supports Embedded SDIO-ATA standard UART.

5.12.4.29 #define CY_U3P_SDIO_INTFC_GPS (0x04)

Function supports SDIO GPS standard interface.

5.12.4.30 #define CY_U3P_SDIO_INTFC_NONE (0x00)

SDIO Standard Function Interface Codes (as defined by SDIO 3.0 Spec).

No standard Interface supported by the function

5.12.4.31 #define CY_U3P_SDIO_INTFC_PHS (0x06)

Function supports SDIO PHS standard interface.

5.12.4.32 #define CY_U3P_SDIO_INTFC_UART (0x01)

Function supports SDIO standard UART.

5.12.4.33 #define CY_U3P_SDIO_INTFC_WLAN (0x07)

Function supports SDIO WLAN standard interface.

5.12.4.34 #define CY_U3P_SDIO_LOW_SPEED (0x00)

Speed of the SDIO Cards.Low speed.

5.12.4.35 #define CY_U3P_SDIO_READ_AFTER_WRITE (0x01)

Read After Write flag for CMD52

5.12.4.36 #define CY_U3P_SDIO_REG_BUS_INTERFACE_CONTROL (0x07)

SDIO bus interface control register address.

5.12.4.37 #define CY_U3P_SDIO_REG_BUS_SUSPEND (0x0C)

SDIO bus suspend register address.

5.12.4.38 #define CY_U3P_SDIO_REG_CARD_CAPABILITY (0x08)

SDIO card capability register address.

5.12.4.39 #define CY_U3P_SDIO_REG_CCCR_HIGH_SPEED (0x13)

Bus speed select register address.

5.12.4.40 #define CY_U3P_SDIO_REG_CCCR_REVISION (0x00)

Address for various bytes in the CCCR.

CCCR revision register address.

5.12.4.41 #define CY_U3P_SDIO_REG_CIS_PTR_D0 (0x09)

CIS pointer: First byte.

5.12.4.42 #define CY_U3P_SDIO_REG_CIS_PTR_D1 (0x0A)

CIS pointer: Second byte.

5.12.4.43 #define CY_U3P_SDIO_REG_CIS_PTR_D2 (0x0B)

CIS pointer: Third byte.

5.12.4.44 #define CY_U3P_SDIO_REG_DRIVER_STRENGTH (0x15)

Driver strength register address.

5.12.4.45 #define CY_U3P_SDIO_REG_EXEC_FLAGS (0x0E)

Exec flags register address.

5.12.4.46 #define CY_U3P_SDIO_REG_FBR_CIS_PTR_D1 CY_U3P_SDIO_REG_CIS_PTR_D1

CIS Pointer.

5.12.4.47 #define CY_U3P_SDIO_REG_FBR_CIS_PTR_D2 CY_U3P_SDIO_REG_CIS_PTR_D2

CIS Pointer.

5.12.4.48 #define CY_U3P_SDIO_REG_FBR_CIS_PTR_DO CY_U3P_SDIO_REG_CIS_PTR_D0

CIS Pointer.

5.12.4.49 #define CY_U3P_SDIO_REG_FBR_CSA_PTR_D1 (0x0D)

CSA pointer.

5.12.4.50 #define CY_U3P_SDIO_REG_FBR_CSA_PTR_D2 (0x0E)

CSA pointer.

5.12.4.51 #define CY_U3P_SDIO_REG_FBR_CSA_PTR_D0 (0x0C)

CSA pointer.

5.12.4.52 #define CY_U3P_SDIO_REG_FBR_DATA_ACCESS_WINDOW (0x0F)

Data access window to CSA.

5.12.4.53 #define CY_U3P_SDIO_REG_FBR_EXT_INTERFACE_CODE (0x01)

Extended SDIO function interface code.

5.12.4.54 #define CY_U3P_SDIO_REG_FBR_INTERFACE_CODE (0x00)

Address for various bytes in the FBR.
Standard function interface code.

5.12.4.55 #define CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D0 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D0

Block size.

5.12.4.56 #define CY_U3P_SDIO_REG_FBR_IO_BLOCKSIZE_D1 CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D1

Block size.

5.12.4.57 #define CY_U3P_SDIO_REG_FBR_POWER_SELECT (0x02)

Power selection.

5.12.4.58 #define CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D0 (0x10)

Function 0 block size: First byte.

5.12.4.59 #define CY_U3P_SDIO_REG_FUNCTION_BLOCKSIZE_D1 (0x11)

Function 0 block size: Second byte.

5.12.4.60 #define CY_U3P_SDIO_REG_FUNCTION_SELECT (0x0D)

Function select register address.

5.12.4.61 #define CY_U3P_SDIO_REG_INTERRUPT_EXTENSION (0x16)

Interrupt extension register address.

5.12.4.62 #define CY_U3P_SDIO_REG_IO_ABORT (0x06)

IO abort register address.

5.12.4.63 #define CY_U3P_SDIO_REG_IO_ENABLE (0x02)

IO Enable register address.

5.12.4.64 #define CY_U3P_SDIO_REG_IO_INTR_ENABLE (0x04)

IO interrupt enable register address.

5.12.4.65 #define CY_U3P_SDIO_REG_IO_INTR_PENDING (0x05)

IO interrupt pending register address.

5.12.4.66 #define CY_U3P_SDIO_REG_IO_READY (0x03)

IO ready register address.

5.12.4.67 #define CY_U3P_SDIO_REG_POWER_CONTROL (0x12)

Power control register address.

5.12.4.68 #define CY_U3P_SDIO_REG_READY_FLAGS (0x0F)

Ready flags register address.

5.12.4.69 #define CY_U3P_SDIO_REG_SD_SPEC_REVISION (0x01)

SD Spec revision register address.

5.12.4.70 #define CY_U3P_SDIO_REG_UHS_I_SUPPORT (0x14)

UHS-I support register address.

5.12.4.71 #define CY_U3P_SDIO_RESET (0x08)

SDIO reset flag.

5.12.4.72 #define CY_U3P_SDIO_SAI (0x01)

SDIO Interrupt Extension flags.

Support Asynchronous Interrupt.

5.12.4.73 #define CY_U3P_SDIO_SD_Version_1_00 (0x00)

SDIO SD Physical Layer Specifications.

SD physical layer version 1.00

5.12.4.74 #define CY_U3P_SDIO_SD_Version_1_10 (0x10)

SD physical layer version 1.10

5.12.4.75 #define CY_U3P_SDIO_SD_Version_2_00 (0x20)

SD physical layer version 2.00

5.12.4.76 #define CY_U3P_SDIO_SD_Version_3_00 (0x30)

SD physical layer version 3.00

5.12.4.77 #define CY_U3P_SDIO_SDDR50_SPEED (0x80)

DDR50.

5.12.4.78 #define CY_U3P_SDIO_SSDR104_SPEED (0x40)

SDR104.

5.12.4.79 #define CY_U3P_SDIO_SSDR12_SPEED (0x04)

SDR12.

5.12.4.80 #define CY_U3P_SDIO_SSDR25_SPEED (0x10)

SDR25.

5.12.4.81 #define CY_U3P_SDIO_SSDR50_SPEED (0x20)

SDR50.

5.12.4.82 #define CY_U3P_SDIO_SUPPORT_HIGH_SPEED (0x01)

High Speed support flag.

5.12.4.83 #define CY_U3P_SDIO_UHS_SDDR50 (0x04)

DDR50 support.

5.12.4.84 #define CY_U3P_SDIO_UHS_SSDR104 (0x02)

SDR104 support.

5.12.4.85 #define CY_U3P_SDIO_UHS_SSDR50 (0x01)

SDIO UHS-I support flags.
SDR50 support.

5.12.4.86 #define CY_U3P_SDIO_Version_1_00 (0x00)

SDIO Specification Revision Numbers.
SDIO version 1.00

5.12.4.87 #define CY_U3P_SDIO_Version_1_10 (0x01)

SDIO version 1.10

5.12.4.88 #define CY_U3P_SDIO_Version_1_20 (0x02)

SDIO version 1.20

5.12.4.89 #define CY_U3P_SDIO_Version_2_00 (0x03)

SDIO version 2.00

5.12.4.90 #define CY_U3P_SDIO_Version_3_00 (0x04)

SDIO version 3.00

5.13 firmware/u3p_firmware/inc/cyu3descriptor.h File Reference

DMA descriptor management.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PDmaDescriptor_t

  Descriptor data structure.

Macros

- #define CY_U3P_DMA_DSCR0_LOCATION (0x40000000)

  This is the location of the descriptor pool. As the DMA descriptors are fixed in the FX3 memory map by the device architecture, this definition should not be changed.
- #define CY_U3P_DMA_DSCR_COUNT (512)

  This is the number of descriptors in the pool.
- #define CY_U3P_DMA_DSCR_SIZE (16)

  This is the size of each descriptor in bytes. This value is defined by the FX3 device architecture.

Typedefs

- typedef struct CyU3PDmaDescriptor_t CyU3PDmaDescriptor_t

  Descriptor data structure.

Functions

- void CyU3PDmaDscrListCreate (void)

  Create and initialize the free descriptor list.
- void CyU3PDmaDscrListDestroy (void)

Destroy the free descriptor list.

- CyU3PReturnStatus_t CyU3PDmaDscrGet (uint16_t *index_p)

  Get a descriptor number from the free list.
- CyU3PReturnStatus_t CyU3PDmaDscrPut (uint16_t index)

  Add a descriptor number to the free list.
- uint16_t CyU3PDmaDscrGetFreeCount (void)

  Get the number of free descriptors available.
- CyU3PReturnStatus_t CyU3PDmaDscrSetConfig (uint16_t index, CyU3PDmaDescriptor_t *dscr_p)

  Update the DMA descriptor configuration.
- CyU3PReturnStatus_t CyU3PDmaDscrGetConfig (uint16_t index, CyU3PDmaDescriptor_t *dscr_p)

  Get the descriptor configuration.
- CyU3PReturnStatus_t CyU3PDmaDscrChainCreate (uint16_t *dscrIndex_p, uint16_t count, uint16_t bufferSize, uint32_t dscrSync)

  Create a circular chain of descriptors.
- void CyU3PDmaDscrChainDestroy (uint16_t dscrIndex, uint16_t count, CyBool_t isProdChain, CyBool_t freeBuffer)

  Frees the previously created chain of descriptors.

Variables

- CyU3PDmaDescriptor_t * **glDmaDescriptor**

5.13.1 Detailed Description

DMA descriptor management. **Summary**

DMA descriptors are data structures that keep track of the source/destinations and the memory buffers for a data transfer through the FX3 device. The firmware maintains a queue of DMA descriptors that are allocated as required and used by the corresponding firmware modules.

This file defines the data structures and the interfaces for descriptor management.

5.13.2 Descriptor Functions

This section documents the utility functions that work with the DMA descriptors. These functions are supposed to be internal functions for the FX3 API library's use and are not expected to be called directly by the user application.

If an application chooses to call these functions, extreme care must be taken to validate the parameters being passed; as these functions do not perform any error checks. Passing in incorrect/invalid parameters can result in unpredictable behavior. In particular, the buffer address in the DMA descriptor needs to be in system memory area. Any other buffer address can cause the entire DMA engine to freeze, requiring a full device reset.

5.13.3 Typedef Documentation

5.13.3.1 typedef struct CyU3PDmaDescriptor_t CyU3PDmaDescriptor_t

Descriptor data structure.

Description

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

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

buffer: (CY_U3P_BUFFER_ADDR_MASK)

sync: (CY_U3P_EN_PROD_INT | CY_U3P_EN_PROD_EVENT | CY_U3P_PROD_IP_MASK | CY_U3P_PROD_-SCK_MASK | CY_U3P_EN_CONS_INT | CY_U3P_EN_CONS_EVENT | CY_U3P_CONS_IP_MASK | CY_U3P_-CONS_SCK_MASK)

chain: (CY_U3P_WR_NEXT_DSCR_MASK | CY_U3P_RD_NEXT_DSCR_MASK)

size: (CY_U3P_BYTE_COUNT_MASK | CY_U3P_BUFFER_SIZE_MASK | CY_U3P_BUFFER_OCCUPIED | CY-_U3P_BUFFER_ERROR | CY_U3P_EOP | CY_U3P_MARKER)

See Also

CyU3PDmaDscrGetConfig
CyU3PDmaDscrSetConfig

5.13.4 Function Documentation

5.13.4.1 CyU3PReturnStatus_t CyU3PDmaDscrChainCreate (uint16_t * *dscrIndex_p*, uint16_t *count*, uint16_t *bufferSize*, uint32_t *dscrSync*)

Create a circular chain of descriptors.

Description

The function creates a chain of descriptors for DMA operations. Both the producer and consumer chains are created with the same values. The descriptor sync parameter is updated as provided. A DMA buffer is allocated if the buffer-Size variable is non zero. This function is mainly used by the DMA channel APIs, and can also be used to customize descriptors and do advanced DMA operations.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_MEMORY_ERROR - if descriptor or DMA buffer allocation failed.

CY_U3P_ERROR_BAD_ARGUMENT - if number of descriptors required is zero OR the index is NULL.

See Also

CyU3PDmaDscrChainDestroy

Parameters

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

5.13.4.2 void CyU3PDmaDscrChainDestroy (uint16_t *dscrIndex*, uint16_t *count*, CyBool_t *isProdChain*, CyBool_t *freeBuffer*)

Frees the previously created chain of descriptors.

Description

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

Return value

None

See Also

CyU3PDmaDscrChainDestroy

Parameters

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

5.13.4.3 CyU3PReturnStatus_t CyU3PDmaDscrGet (uint16_t * *index_p*)

Get a descriptor number from the free list.

Description

This function searches the free list for the first available descriptor, and returns the index. This function is used by the DMA channel APIs, and can be used to do advanced DMA programming based on direct socket access.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if a NULL pointer is passed.

CY_U3P_ERROR_FAILURE - if no free descriptor is found.

See Also

CyU3PDmaDscrPut
CyU3PDmaDscrGetFreeCount

Parameters

index_p	Output parameter which is filled with the free descriptor index.

5.13.4.4 CyU3PReturnStatus_t CyU3PDmaDscrGetConfig (uint16_t *index*, CyU3PDmaDescriptor_t * *dscr_p*)

Get the descriptor configuration.

Description

Read the current contents of the specified DMA descriptor into the output descriptor data structure.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if an ivalid index or a NULL pointer is passed.

See Also

CyU3PDmaDscrGetConfig

Parameters

index	Index of the descriptor to read.
dscr_p	Output parameter that will be filled with the descriptor values.

5.13.4.5 uint16_t CyU3PDmaDscrGetFreeCount (void)

Get the number of free descriptors available.

Description

This function returns the number of free descriptors available for use.

Return value

The number of free descriptors available.

See Also

CyU3PDmaDscrGet
CyU3PDmaDscrPut

5.13.4.6 void CyU3PDmaDscrListCreate (void)

Create and initialize the free descriptor list.

Description

This function initializes the free descriptor list, and marks all of the descriptors as available. This function is invoked internal to the library and is not expected to be called explicitly.

Return value

None

See Also

CyU3PDmaDscrListDestroy

5.13.4.7 void CyU3PDmaDscrListDestroy (void)

Destroy the free descriptor list.

Description

This function de-initializes the free descriptor list, and marks all of the descriptors as non-available. This function is invoked internal to the library and should not be called explicitly.

Return value

None

See Also

CyU3PDmaDscrListCreate

5.13.4.8 CyU3PReturnStatus_t CyU3PDmaDscrPut (uint16_t *index*)

Add a descriptor number to the free list.

Description

This function marks a descriptor as available in the free list. This function is used internally by the DMA channel APIs, and can be used to do advanced DMA programming based on direct socket access.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if an invalid index is passed.

See Also

CyU3PDmaDscrGet
CyU3PDmaDscrGetFreeCount

Parameters

index	Descriptor index to be marked free.

5.13.4.9 CyU3PReturnStatus_t CyU3PDmaDscrSetConfig (uint16_t *index*, CyU3PDmaDescriptor_t * *dscr_p*)

Update the DMA descriptor configuration.

Description

Update the specified DMA descriptor in FX3 memory with data from the descriptor structure passed in as parameter.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if an invalid index or a NULL pointer is passed.

See Also

CyU3PDmaDscrGetConfig

Parameters

index	Index of the descriptor to be updated.
dscr_p	Pointer to descriptor structure containing the desired configuration.

5.14 firmware/u3p_firmware/inc/cyu3dma.h File Reference

DMA driver and API definitions for EZ-USB FX3.

```
#include "cyu3os.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PDmaBuffer_t

  DMA buffer data structure.
- union CyU3PDmaCBInput_t

  DMA channel callback input.
- struct CyU3PDmaChannelConfig_t

  DMA channel parameters.
- struct CyU3PDmaMultiChannelConfig_t

  DMA multi-channel parameter structure.
- struct CyU3PDmaChannel

  DMA Channel structure.
- struct CyU3PDmaMultiChannel

  DMA multi-channel structure.

Macros

- #define CY_U3P_DMA_MIN_MULTI_SCK_COUNT (2)

  Macro defining the minimum required sockets for multi socket DMA channels. Since normal DMA channels can be used for single socket, the minimum number supported is 2.
- #define CY_U3P_DMA_MAX_MULTI_SCK_COUNT (4)

  Macro defining the maximum allowed socket for multi socket DMA channels.
- #define CY_U3P_DMA_BUFFER_MARKER (1u << 0)

  Software based marker that can be applied on a DMA buffer. This bit is part of the four bit status flag that is associated with each DMA buffer descriptor. The DMA manager makes use of this bit to indicate that a specific buffer is expected to be discarded.
- #define CY_U3P_DMA_BUFFER_EOP (1u << 1)

  This bit indicates this descriptor refers to the last buffer of a packet or transfer. Packets/transfers may span more than one buffer. The producing IP provides this marker by providing the EOP signal to its DMA adapter.
- #define CY_U3P_DMA_BUFFER_ERROR (1u << 2)

  This bit is part of the 4 bit DMA buffer status field and indicates that the buffer transfer has hit an error.
- #define CY_U3P_DMA_BUFFER_OCCUPIED (1u << 3)

  This status bit indicates the buffer has valid data. This bit can be set even if the buffer count is zero, because the device needs to handle Zero length packets as well.
- #define CY_U3P_DMA_BUFFER_STATUS_MASK (0x000F)

  Mask to retrieve all DMA buffer status bits.
- #define CY_U3P_DMA_BUFFER_STATUS_WRITE_MASK (0x000E)

  Mask to identify status bits that can be modified by the user application. Bit 0 (MARKER) is reserved for internal use by the DMA manager.
- #define CY_U3P_DMA_BUFFER_AREA_BASE (uint8_t *)(0x40000000)

  Start address of the allowed buffer memory area for DMA operations. Corresponds to the beginning of the FX3 System RAM.
- #define CY_U3P_DMA_BUFFER_AREA_LIMIT (uint8_t *)(0x40080000)

  End address of the allowed buffer memory area for DMA operations. Corresponds to the end of the System RAM for the CYUSB3014 device.
- #define CY_U3P_DMA_MAX_BUFFER_SIZE (0xFFF0)

  Maximum allowed size for a single DMA buffer. This is defined by the FX3 device architecture.
- #define CY_U3P_DMA_MAX_AVAIL_COUNT (31)

  The maximum number of available buffers before a DMA channel can receive data from the producer. This is the maximum value that the prodAvailCount can be set to.

Typedefs

- typedef enum CyU3PDmaSocketId_t CyU3PDmaSocketId_t

  DMA socket IDs for all sockets in the device.
- typedef enum CyU3PDmaType_t CyU3PDmaType_t

  List of DMA single channel types.
- typedef enum CyU3PDmaMultiType_t CyU3PDmaMultiType_t

  List of DMA multi-channel types.
- typedef enum CyU3PDmaState_t CyU3PDmaState_t

  List of different DMA channel states.
- typedef enum CyU3PDmaMode_t CyU3PDmaMode_t

  List of different DMA transfer modes.
- typedef enum CyU3PDmaCbType_t CyU3PDmaCbType_t

  List of DMA channel callback types.
- typedef enum CyU3PDmaSckSuspType_t CyU3PDmaSckSuspType_t

  List of DMA socket suspend options.
- typedef struct CyU3PDmaBuffer_t CyU3PDmaBuffer_t

DMA buffer data structure.

- typedef union CyU3PDmaCBInput_t CyU3PDmaCBInput_t

  DMA channel callback input.
- typedef struct CyU3PDmaChannel **CyU3PDmaChannel**
- typedef struct CyU3PDmaMultiChannel **CyU3PDmaMultiChannel**
- typedef void(∗ CyU3PDmaCallback_t)(CyU3PDmaChannel ∗handle, CyU3PDmaCbType_t type, CyU3PDmaCBInput_t ∗input)

  DMA channel callback function type.
- typedef void(∗ CyU3PDmaMultiCallback_t)(CyU3PDmaMultiChannel ∗handle, CyU3PDmaCbType_t type, CyU3PDmaCBInput_t ∗input)

  Multi socket DMA channel callback function type.
- typedef struct CyU3PDmaChannelConfig_t CyU3PDmaChannelConfig_t

  DMA channel parameters.
- typedef struct CyU3PDmaMultiChannelConfig_t CyU3PDmaMultiChannelConfig_t

  DMA multi-channel parameter structure.

Enumerations

- enum CyU3PDmaSocketId_t {
  CY_U3P_LPP_SOCKET_I2S_LEFT = 0x0000, CY_U3P_LPP_SOCKET_I2S_RIGHT, CY_U3P_LPP_SOCKET_I2C_CONS, CY_U3P_LPP_SOCKET_UART_CONS,
  CY_U3P_LPP_SOCKET_SPI_CONS, CY_U3P_LPP_SOCKET_I2C_PROD, CY_U3P_LPP_SOCKET_UART_PROD, CY_U3P_LPP_SOCKET_SPI_PROD,
  CY_U3P_PIB_SOCKET_0 = 0x0100, CY_U3P_PIB_SOCKET_1, CY_U3P_PIB_SOCKET_2, CY_U3P_PIB_SOCKET_3,
  CY_U3P_PIB_SOCKET_4, CY_U3P_PIB_SOCKET_5, CY_U3P_PIB_SOCKET_6, CY_U3P_PIB_SOCKET_7,
  CY_U3P_PIB_SOCKET_8, CY_U3P_PIB_SOCKET_9, CY_U3P_PIB_SOCKET_10, CY_U3P_PIB_SOCKET_11,
  CY_U3P_PIB_SOCKET_12, CY_U3P_PIB_SOCKET_13, CY_U3P_PIB_SOCKET_14, CY_U3P_PIB_SOCKET_15,
  CY_U3P_PIB_SOCKET_16, CY_U3P_PIB_SOCKET_17, CY_U3P_PIB_SOCKET_18, CY_U3P_PIB_SOCKET_19,
  CY_U3P_PIB_SOCKET_20, CY_U3P_PIB_SOCKET_21, CY_U3P_PIB_SOCKET_22, CY_U3P_PIB_SOCKET_23,
  CY_U3P_PIB_SOCKET_24, CY_U3P_PIB_SOCKET_25, CY_U3P_PIB_SOCKET_26, CY_U3P_PIB_SOCKET_27,
  CY_U3P_PIB_SOCKET_28, CY_U3P_PIB_SOCKET_29, CY_U3P_PIB_SOCKET_30, CY_U3P_PIB_SOCKET_31,
  CY_U3P_SIB_SOCKET_0 = 0x0200, CY_U3P_SIB_SOCKET_1, CY_U3P_SIB_SOCKET_2, CY_U3P_SIB_SOCKET_3,
  CY_U3P_SIB_SOCKET_4, CY_U3P_SIB_SOCKET_5, 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 }

DMA socket IDs for all sockets in the device.

- 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 }

  List of DMA single channel types.
- 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 }

  List of DMA multi-channel types.
- 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 }

  List of different DMA channel states.
- enum CyU3PDmaMode_t { CY_U3P_DMA_MODE_BYTE = 0, CY_U3P_DMA_MODE_BUFFER, CY_U3P_DMA_NUM_MODES }

  List of different DMA transfer modes.
- enum CyU3PDmaCbType_t {
CY_U3P_DMA_CB_XFER_CPLT = (1 << 0), CY_U3P_DMA_CB_SEND_CPLT = (1 << 1), CY_U3P_DMA_CB_RECV_CPLT = (1 << 2), CY_U3P_DMA_CB_PROD_EVENT = (1 << 3),
CY_U3P_DMA_CB_CONS_EVENT = (1 << 4), CY_U3P_DMA_CB_ABORTED = (1 << 5), CY_U3P_DMA_CB_ERROR = (1 << 6), CY_U3P_DMA_CB_PROD_SUSP = (1 << 7),
CY_U3P_DMA_CB_CONS_SUSP = (1 << 8) }

  List of DMA channel callback types.
- enum CyU3PDmaSckSuspType_t { CY_U3P_DMA_SCK_SUSP_NONE = 0, CY_U3P_DMA_SCK_SUSP_EOP, CY_U3P_DMA_SCK_SUSP_CUR_BUF, CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF }

  List of DMA socket suspend options.

Functions

- CyU3PDmaChannel * CyU3PDmaChannelGetHandle (CyU3PDmaSocketId_t sckId)

  Fetch a handle to the DMA channel corresponding to the specified socket.
- CyU3PReturnStatus_t CyU3PDmaChannelCreate (CyU3PDmaChannel *handle, CyU3PDmaType_t type, CyU3PDmaChannelConfig_t *config)

  Create a one-to-one socket DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelDestroy (CyU3PDmaChannel *handle)

  Destroy a one-to-one socket DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelUpdateMode (CyU3PDmaChannel *handle, CyU3PDmaMode_t dmaMode)

  Update the DMA mode for a one-to-one DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelSetXfer (CyU3PDmaChannel *handle, uint32_t count)

  Setup a one-to-one DMA channel for data transfer.
- CyU3PReturnStatus_t CyU3PDmaChannelSetWrapUp (CyU3PDmaChannel *handle)

Wraps up the current active buffer for the channel from the producer side.

- CyU3PReturnStatus_t CyU3PDmaChannelSetSuspend (CyU3PDmaChannel *handle, CyU3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp)

  Set the suspend options for the sockets associated with a DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelResume (CyU3PDmaChannel *handle, CyBool_t isProdResume, CyBool_t isConsResume)

  Resume a suspended DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelAbort (CyU3PDmaChannel *handle)

  Aborts a DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelReset (CyU3PDmaChannel *handle)

  Aborts and resets a DMA channel.
- CyU3PReturnStatus_t CyU3PDmaChannelSetupSendBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p)

  Send the contents of a user provided buffer to the consumer.
- CyU3PReturnStatus_t CyU3PDmaChannelSetupRecvBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p)

  Receive data into a user provided DMA buffer.
- CyU3PReturnStatus_t CyU3PDmaChannelWaitForRecvBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)

  Wait until an override read operation is completed.
- CyU3PReturnStatus_t CyU3PDmaChannelGetBuffer (CyU3PDmaChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)

  Get the current buffer pointer.
- CyU3PReturnStatus_t CyU3PDmaChannelCommitBuffer (CyU3PDmaChannel *handle, uint16_t count, uint16_t bufStatus)

  Commit a data buffer to be sent to the consumer.
- CyU3PReturnStatus_t CyU3PDmaChannelDiscardBuffer (CyU3PDmaChannel *handle)

  Drop a data buffer without sending out to the consumer.
- CyU3PReturnStatus_t CyU3PDmaChannelWaitForCompletion (CyU3PDmaChannel *handle, uint32_t waitOption)

  Wait for the current DMA transaction to complete.
- CyU3PReturnStatus_t CyU3PDmaChannelGetStatus (CyU3PDmaChannel *handle, CyU3PDmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount)

  This function returns the current channel status.
- CyU3PReturnStatus_t CyU3PDmaChannelCacheControl (CyU3PDmaChannel *handle, CyBool_t isDmaHandleDCache)

  This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.
- CyU3PDmaMultiChannel * CyU3PDmaMultiChannelGetHandle (CyU3PDmaSocketId_t sckId)

  Identifies the multi-channel that is associated with the specified socket.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelCreate (CyU3PDmaMultiChannel *handle, CyU3PDmaMultiType_t type, CyU3PDmaMultiChannelConfig_t *config)

  Create a multi-socket DMA channel with the specified parameters.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelDestroy (CyU3PDmaMultiChannel *handle)

  Destroy a multi-socket DMA channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelUpdateMode (CyU3PDmaMultiChannel *handle, CyU3PDmaMode_t dmaMode)

  Update the DMA mode for a multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelSetXfer (CyU3PDmaMultiChannel *handle, uint32_t count, uint16_t multiSckOffset)

  Prepare a DMA multi-channel for data transfer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelSetWrapUp (CyU3PDmaMultiChannel *handle, uint16_t multiSckOffset)

Wraps up the current active buffer on the producer socket.

- CyU3PReturnStatus_t CyU3PDmaMultiChannelSetSuspend (CyU3PDmaMultiChannel *handle, CyU3PDmaSckSuspType_t prodSusp, CyU3PDmaSckSuspType_t consSusp)

  Update the suspend options for the sockets associated with a DMA multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelResume (CyU3PDmaMultiChannel *handle, CyBool_t isProdResume, CyBool_t isConsResume)

  Resume a suspended DMA multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelAbort (CyU3PDmaMultiChannel *handle)

  Abort the ongoing transfer on a DMA multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelReset (CyU3PDmaMultiChannel *handle)

  Abort ongoing transfers on and resets a DMA multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupSendBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset)

  Send the contents of a user provided buffer to the consumer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupRecvBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint16_t multiSckOffset)

  Receive data into a user provided DMA buffer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForRecvBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)

  Wait until an override read operation is completed.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelGetBuffer (CyU3PDmaMultiChannel *handle, CyU3PDmaBuffer_t *buffer_p, uint32_t waitOption)

  Get the current buffer pointer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelCommitBuffer (CyU3PDmaMultiChannel *handle, uint16_t count, uint16_t bufStatus)

  Commit the buffer to be sent to the consumer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelDiscardBuffer (CyU3PDmaMultiChannel *handle)

  Drop a data buffer without sending out to the consumer.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForCompletion (CyU3PDmaMultiChannel *handle, uint32_t waitOption)

  Wait for the current DMA transaction to complete on a DMA multi-channel.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelGetStatus (CyU3PDmaMultiChannel *handle, CyU3PDmaState_t *state, uint32_t *prodXferCount, uint32_t *consXferCount, uint8_t sckIndex)

  This functions returns the current multi-channel status.
- CyU3PReturnStatus_t CyU3PDmaMultiChannelCacheControl (CyU3PDmaMultiChannel *handle, CyBool_t isDmaHandleDCache)

  This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.
- void CyU3PDmaEnableMulticast (void)

  Enable creation and management of DMA multicast channels.
- CyU3PReturnStatus_t CyU3PDmaMulticastSocketSelect (CyU3PDmaMultiChannel *chHandle, uint32_t consMask)

  Select the active consumers on a multicast DMA channel.

5.14.1 Detailed Description

DMA driver and API definitions for EZ-USB FX3. **Description**

The FX3 device architecture includes a distributed DMA fabric that allows high performance data transfer between any of the external interfaces of the device. This DMA engine is a custom implementation that makes use of multiple registers and data structures for the data path management.

The DMA driver in the FX3 firmware takes care of configuring the DMA data paths as desired and performs all runtime control operations such as interrupt handling. The FX3 library also provides a set of APIs through which the user application can set up and control DMA data paths.

5.14.2 Typedef Documentation

5.14.2.1 typedef struct CyU3PDmaBuffer_t CyU3PDmaBuffer_t

DMA buffer data structure.

Description

The data structure is used to describe the status of a DMA buffer. It holds the address, size, valid data count, and the status information. This type is used in DMA callbacks and APIs to identify the properties of the data buffer to be transferred.

See Also

CyU3PDmaCBInput_t
CyU3PDmaChannelGetBuffer
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForRecvBuffer
CyU3PDmaMultiChannelGetBuffer
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForRecvBuffer

5.14.2.2 typedef void(* CyU3PDmaCallback_t)(CyU3PDmaChannel *handle,CyU3PDmaCbType_t type,CyU3PDmaCBInput_t *input)

DMA channel callback function type.

Description

The FX3 DMA manager supports a number of event notifications that can be sent to the user application during system operation. These event notifications are provided per DMA channel, and the types of events required can be selected at the time of channel creation.

This type defines the prototype for the callback function which will be invoked to provide these event notifications.

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

If produce events are enabled, the application is expected to be fast enough to handle the incoming data rate. The input parameter can be stale if the handling of the buffer is not done in a timely fashion.

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

In case of MANUAL or MANUAL_IN channels, the input parameter points to the first buffer left to be committed to the consumer socket. If the buffer is not committed before the next callback, then the input parameter shall be stale data.

See Also

CyU3PDmaCBInput_t
CyU3PDmaChannelConfig_t
CyU3PDmaChannelCreate

5.14.2.3 typedef union CyU3PDmaCBInput_t CyU3PDmaCBInput_t

DMA channel callback input.

Description

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

See Also

CyU3PDmaBuffer_t
CyU3PDmaCallback_t

5.14.2.4 typedef enum CyU3PDmaCbType_t CyU3PDmaCbType_t

List of DMA channel callback types.

Description

This type lists the various callback types that are supported by the DMA manager for various DMA channels. The user can register the combination of these events for which event notifications are required at channel creation time.

See Also

CyU3PDmaCallback_t
CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaMultiChannelCreate

5.14.2.5 typedef struct CyU3PDmaChannelConfig_t CyU3PDmaChannelConfig_t

DMA channel parameters.

Description

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

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

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

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

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

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

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

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

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

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

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

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

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

See Also

CyU3PDmaChannelCreate

5.14.2.6 typedef enum CyU3PDmaMode_t CyU3PDmaMode_t

List of different DMA transfer modes.

Description

The following are the different types of DMA transfer modes. The default mode of operation is byte mode. The buffer mode is useful only when there are variable data sized transfers and when firmware handling is required after a finite number of buffers.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaChannelUpdateMode
CyU3PDmaMultiChannelUpdateMode

5.14.2.7 typedef void(∗ CyU3PDmaMultiCallback_t)(CyU3PDmaMultiChannel ∗handle,CyU3PDmaCbType_t type,CyU3PDmaCBInput_t ∗input)

Multi socket DMA channel callback function type.

Description

Callback function type that is associated with DMA multi-channels. All of the usage restrictions and guidelines that apply to normal DMA channel callbacks (CyU3PDmaCallback_t) also apply to these callbacks. The only difference is in the type of the first parameter passed to the callback function.

See Also

CyU3PDmaCallback_t
CyU3PDmaCBInput_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaMultiChannelCreate

5.14.2.8 typedef struct CyU3PDmaMultiChannelConfig_t CyU3PDmaMultiChannelConfig_t

DMA multi-channel parameter structure.

Description

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

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

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

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

The size, count, prodHeader, prodFooter, consHeader and prodAvailCount fields are used in the same way as in the CyU3PDmaChannelConfig_t structure.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelCreate

5.14.2.9 typedef enum CyU3PDmaMultiType_t CyU3PDmaMultiType_t

List of DMA multi-channel types.

Description

In some cases, a simple one-to-one DMA channel is insufficient to handle the data flowing through the FX3 device. For example, a user may need to use multiple PIB (GPIF) sockets to collect the data coming from an image sensor without any data loss. Special handling is required to combine the data from these sockets into a single USB endpoint for streaming.

Such special DMA handling is achieved through the use of multi-channels. Different types of multi-channels are supported by the DMA manager, and this enumerated type lists them. Special APIs with a CyU3PDmaMultiChannel prefix need to be used for creating and managing DMA multi-channels.

See Also

CyU3PDmaType_t
CyU3PDmaMultiChannelCreate
CyU3PDmaEnableMulticast

5.14.2.10 typedef enum CyU3PDmaSckSuspType_t CyU3PDmaSckSuspType_t

List of DMA socket suspend options.

Description

The producer and consume sockets associated with a DMA channel can be suspended based on various triggers. Each of these suspend triggers/options behaves differently.

Note

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

See Also

CyU3PDmaChannelCreate
CyU3PDmaChannelSetSuspend
CyU3PDmaChannelResume
CyU3PDmaMultiChannelCreate

5.14.2.11 typedef enum CyU3PDmaSocketId_t CyU3PDmaSocketId_t

DMA socket IDs for all sockets in the device.

Description

This is a software representation of all sockets on the device. The socket ID has two parts: IP number and socket number. Each peripheral (IP) has a fixed ID. LPP is 0, PIB is 1, Storage port is 2, USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and have to be used as defined. The P-IB sockets 0-15 can be used as both producer and consumer, but the PIB sockets 16-31 are strictly producer sockets. The UIB sockets are defined as 0-15 producer and 0-15 consumer sockets. The CPU sockets are virtual representations and have no backing hardware block.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaMultiChannelCreate

5.14.2.12 typedef enum CyU3PDmaState_t CyU3PDmaState_t

List of different DMA channel states.

Description

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

See Also

CyU3PDmaChannelGetStatus

5.14.2.13 typedef enum CyU3PDmaType_t CyU3PDmaType_t

List of DMA single channel types.

Description

A DMA channel aggregates all of the resources required to manage a unique data flow through the FX3 device. A DMA channel needs to be created before any DMA operation can be performed.

In the normal case, each DMA channel has one producer socket through which data is received on the FX3 device; and one consumer socket through which data is sent out of the FX3 device. The DMA channels can be classified into multiple types based on the amount of firmware intervention and control that is required in the data flow.

This enumeration lists the different types of DMA single (one to one) channels. All APIs that operate on these channels have have a CyU3PDmaChannel prefix.

See Also

CyU3PDmaChannelCreate

5.14.3 Enumeration Type Documentation

5.14.3.1 enum CyU3PDmaCbType_t

List of DMA channel callback types.

Description

This type lists the various callback types that are supported by the DMA manager for various DMA channels. The user can register the combination of these events for which event notifications are required at channel creation time.

See Also

CyU3PDmaCallback_t
CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaMultiChannelCreate

Enumerator

CY_U3P_DMA_CB_XFER_CPLT Transfer has been completed. This event is generated when a finite transfer queued with SetXfer is completed.

CY_U3P_DMA_CB_SEND_CPLT SendBuffer call has been completed. This event is generated when the data queued with SendBuffer has been successfully sent.

CY_U3P_DMA_CB_RECV_CPLT ReceiverBuffer call has been completed. This event is generated when data is received successfully on the producer socket.

CY_U3P_DMA_CB_PROD_EVENT Buffer received from producer. This event is generated when a buffer is generated by the producer socket when a transfer is queued with SetXfer.

CY_U3P_DMA_CB_CONS_EVENT Buffer consumed by the consumer. This event is generated when a buffer is sent out by the consumer socket when a transfer is queued with SetXfer.

CY_U3P_DMA_CB_ABORTED This event is generated when the Abort API is invoked.

CY_U3P_DMA_CB_ERROR This event is generated when the hardware detects an error.

CY_U3P_DMA_CB_PROD_SUSP This event is generated when the producer socket is suspended.

CY_U3P_DMA_CB_CONS_SUSP This event is generated when the consumer socket is suspended.

5.14.3.2 enum CyU3PDmaMode_t

List of different DMA transfer modes.

Description

The following are the different types of DMA transfer modes. The default mode of operation is byte mode. The buffer mode is useful only when there are variable data sized transfers and when firmware handling is required after a finite number of buffers.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaChannelUpdateMode
CyU3PDmaMultiChannelUpdateMode

Enumerator

CY_U3P_DMA_MODE_BYTE Transfer is based on byte count. This is the default mode of operation. The transfer count is done based on number of bytes received / sent.

CY_U3P_DMA_MODE_BUFFER Transfer is based on buffer count. The transfer count is based on the number of buffers generated or consumed. This is useful only when the data size is variable but there should be finite handling after N buffers.

CY_U3P_DMA_NUM_MODES Count of DMA modes. This is just a place holder and not a valid mode.

5.14.3.3 enum CyU3PDmaMultiType_t

List of DMA multi-channel types.

Description

In some cases, a simple one-to-one DMA channel is insufficient to handle the data flowing through the FX3 device. For example, a user may need to use multiple PIB (GPIF) sockets to collect the data coming from an image sensor without any data loss. Special handling is required to combine the data from these sockets into a single USB endpoint for streaming.

Such special DMA handling is achieved through the use of multi-channels. Different types of multi-channels are supported by the DMA manager, and this enumerated type lists them. Special APIs with a CyU3PDmaMultiChannel prefix need to be used for creating and managing DMA multi-channels.

See Also

CyU3PDmaType_t
CyU3PDmaMultiChannelCreate
CyU3PDmaEnableMulticast

Enumerator

CY_U3P_DMA_TYPE_AUTO_MANY_TO_ONE Auto mode many to one interleaved DMA channel.

CY_U3P_DMA_TYPE_AUTO_ONE_TO_MANY Auto mode one to many interleaved DMA channel.

CY_U3P_DMA_TYPE_MANUAL_MANY_TO_ONE Manual mode many to one interleaved DMA channel.

CY_U3P_DMA_TYPE_MANUAL_ONE_TO_MANY Manual mode one to many interleaved DMA channel.

CY_U3P_DMA_TYPE_MULTICAST Multicast mode with one producer and multiple consumers for the same buffer. This is a manual channel. Please note that the CyU3PDmaEnableMulticast API needs to be called before any multicast DMA channels are created.

CY_U3P_DMA_NUM_TYPES Number of DMA channel types.

5.14.3.4 enum CyU3PDmaSckSuspType_t

List of DMA socket suspend options.

Description

The producer and consume sockets associated with a DMA channel can be suspended based on various triggers. Each of these suspend triggers/options behaves differently.

Note

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

See Also

CyU3PDmaChannelCreate
CyU3PDmaChannelSetSuspend
CyU3PDmaChannelResume
CyU3PDmaMultiChannelCreate

Enumerator

CY_U3P_DMA_SCK_SUSP_NONE Socket will not be suspended. This option is used to remove any existing suspend triggers on the selected socket.

CY_U3P_DMA_SCK_SUSP_EOP Socket will be suspended after handling any buffer with the EOP bit set. Typically the EOP bit will be set on any data buffers that are wrapped up on the PIB (GPIF) side through a COMMIT action. The DMA buffer with the EOP bit set would have been handled before the socket gets suspended, meaning that it is not possible to make changes to the contents of that data packet. This option can be applied to both producer and consumer sockets, and is sticky.

CY_U3P_DMA_SCK_SUSP_CUR_BUF Socket will be suspended after the current buffer is completed. This can be used to suspend the socket at a defined point to be resumed later. This option is is valid for only the current buffer, and the socket option will change back to CY_U3P_DMA_SCK_SUSP_NONE on resumption.

CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF This option is valid only for consumer sockets, and allows the socket to be suspended before handling any partially filled (count $<$ size) DMA buffer. This mode allows the user to make changes to the data before sending it out. Please note that the suspend option for the socket has to be changed to CY_U3P_DMA_SCK_SUSP_CUR_BUF or CY_U3P_DMA_SCK_SUSP_NONE when resuming the channel operation.

5.14.3.5 enum CyU3PDmaSocketId_t

DMA socket IDs for all sockets in the device.

Description

This is a software representation of all sockets on the device. The socket ID has two parts: IP number and socket number. Each peripheral (IP) has a fixed ID. LPP is 0, PIB is 1, Storage port is 2, USB egress is 3 and USB ingress is 4.

Each peripheral has a number of sockets. The LPP sockets are fixed and have to be used as defined. The PIB sockets 0-15 can be used as both producer and consumer, but the PIB sockets 16-31 are strictly producer sockets. The UIB sockets are defined as 0-15 producer and 0-15 consumer sockets. The CPU sockets are virtual representations and have no backing hardware block.

See Also

CyU3PDmaChannelConfig_t
CyU3PDmaMultiChannelConfig_t
CyU3PDmaChannelCreate
CyU3PDmaMultiChannelCreate

Enumerator

CY_U3P_LPP_SOCKET_I2S_LEFT 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 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_SIB_SOCKET_0 S-port socket number 0.
CY_U3P_SIB_SOCKET_1 S-port socket number 1.
CY_U3P_SIB_SOCKET_2 S-port socket number 2.
CY_U3P_SIB_SOCKET_3 S-port socket number 3.
CY_U3P_SIB_SOCKET_4 S-port socket number 4.
CY_U3P_SIB_SOCKET_5 S-port socket number 5.
CY_U3P_UIB_SOCKET_CONS_0 U-port output socket number 0.
CY_U3P_UIB_SOCKET_CONS_1 U-port output socket number 1.
CY_U3P_UIB_SOCKET_CONS_2 U-port output socket number 2.
CY_U3P_UIB_SOCKET_CONS_3 U-port output socket number 3.
CY_U3P_UIB_SOCKET_CONS_4 U-port output socket number 4.
CY_U3P_UIB_SOCKET_CONS_5 U-port output socket number 5.
CY_U3P_UIB_SOCKET_CONS_6 U-port output socket number 6.
CY_U3P_UIB_SOCKET_CONS_7 U-port output socket number 7.
CY_U3P_UIB_SOCKET_CONS_8 U-port output socket number 8.
CY_U3P_UIB_SOCKET_CONS_9 U-port output socket number 9.
CY_U3P_UIB_SOCKET_CONS_10 U-port output socket number 10.
CY_U3P_UIB_SOCKET_CONS_11 U-port output socket number 11.
CY_U3P_UIB_SOCKET_CONS_12 U-port output socket number 12.
CY_U3P_UIB_SOCKET_CONS_13 U-port output socket number 13.
CY_U3P_UIB_SOCKET_CONS_14 U-port output socket number 14.
CY_U3P_UIB_SOCKET_CONS_15 U-port output socket number 15.
CY_U3P_UIB_SOCKET_PROD_0 U-port input socket number 0.
CY_U3P_UIB_SOCKET_PROD_1 U-port input socket number 1.

CY_U3P_UIB_SOCKET_PROD_2 U-port input socket number 2.

CY_U3P_UIB_SOCKET_PROD_3 U-port input socket number 3.

CY_U3P_UIB_SOCKET_PROD_4 U-port input socket number 4.

CY_U3P_UIB_SOCKET_PROD_5 U-port input socket number 5.

CY_U3P_UIB_SOCKET_PROD_6 U-port input socket number 6.

CY_U3P_UIB_SOCKET_PROD_7 U-port input socket number 7.

CY_U3P_UIB_SOCKET_PROD_8 U-port input socket number 8.

CY_U3P_UIB_SOCKET_PROD_9 U-port input socket number 9.

CY_U3P_UIB_SOCKET_PROD_10 U-port input socket number 10.

CY_U3P_UIB_SOCKET_PROD_11 U-port input socket number 11.

CY_U3P_UIB_SOCKET_PROD_12 U-port input socket number 12.

CY_U3P_UIB_SOCKET_PROD_13 U-port input socket number 13.

CY_U3P_UIB_SOCKET_PROD_14 U-port input socket number 14.

CY_U3P_UIB_SOCKET_PROD_15 U-port input socket number 15.

CY_U3P_CPU_SOCKET_CONS Socket through which the FX3 CPU receives data.

CY_U3P_CPU_SOCKET_PROD Socket through which the FX3 CPU produces data.

5.14.3.6 enum CyU3PDmaState_t

List of different DMA channel states.

Description

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

See Also

CyU3PDmaChannelGetStatus

Enumerator

CY_U3P_DMA_NOT_CONFIGURED DMA channel is unconfigured. This state occurs only when using a stale/uninitialized channel structure. This channel state is set to this when a channel is destroyed.

CY_U3P_DMA_CONFIGURED DMA channel has been configured successfully. The channel reaches this state through the following conditions:

1. Channel is successfully created.
2. Channel is reset.
3. A finite transfer has been successfully completed. A GetStatus call in this case will return a virtual CY_U3P_DMA_XFER_COMPLETED state.
4. One of the override modes have completed successfully. A GetStatus call in this case will return a virtual CY_U3P_DMA_SEND_COMPLETED or CY_U3P_DMA_RECV_COMPLETED state.

CY_U3P_DMA_ACTIVE Channel has active transaction going on. This state is reached when a SetXfer call is invoked and the transfer is on-going.

CY_U3P_DMA_PROD_OVERRIDE The channel is working in producer socket override mode. This state is reached when a SetupSend call is invoked and the transfer is on-going.

CY_U3P_DMA_CONS_OVERRIDE Channel is working in consumer socket override mode. This state is reached when a SetupRecv call is invoked and the transfer is on-going.

CY_U3P_DMA_ERROR Channel has encountered an error. This state is reached when a DMA hardware error is detected.

CY_U3P_DMA_IN_COMPLETION Waiting for all data to drain out. This state is reached when transfer has completed from the producer side, but waiting for the consumer to drain all data.

CY_U3P_DMA_ABORTED The channel is in aborted state. This state is reached when a Abort call is made.

CY_U3P_DMA_NUM_STATES Number of states. This is not a valid state and is just a place holder.

CY_U3P_DMA_XFER_COMPLETED This is a virtual state returned by GetStatus function. The actual state is CY_U3P_DMA_CONFIGURED. This is just the value returned on GetStatus call after completion of finite transfer.

CY_U3P_DMA_SEND_COMPLETED This is a virtual state returned by GetStatus function. The actual state is CY_U3P_DMA_CONFIGURED. This is just the value returned on GetStatus call after completion of producer override mode.

CY_U3P_DMA_RECV_COMPLETED This is a virtual state returned by GetStatus function. The actual state is CY_U3P_DMA_CONFIGURED. This is just the value returned on GetStatus call after completion of consumer override mode.

5.14.3.7 enum CyU3PDmaType_t

List of DMA single channel types.

Description

A DMA channel aggregates all of the resources required to manage a unique data flow through the FX3 device. A DMA channel needs to be created before any DMA operation can be performed.

In the normal case, each DMA channel has one producer socket through which data is received on the FX3 device; and one consumer socket through which data is sent out of the FX3 device. The DMA channels can be classified into multiple types based on the amount of firmware intervention and control that is required in the data flow.

This enumeration lists the different types of DMA single (one to one) channels. All APIs that operate on these channels have have a CyU3PDmaChannel prefix.

See Also

CyU3PDmaChannelCreate

Enumerator

CY_U3P_DMA_TYPE_AUTO Auto mode DMA channel.

CY_U3P_DMA_TYPE_AUTO_SIGNAL Auto mode with 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.

5.14.4 Function Documentation

5.14.4.1 CyU3PReturnStatus_t CyU3PDmaChannelAbort (CyU3PDmaChannel * *handle*)

Aborts a DMA channel.

Description

The function shall abort both the producer and consumer sockets associated with the channel. The data in transition is lost and any active transaction cannot be resumed. This function leaves the channel in an aborted state and requires a reset before the channel can be used again.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelSetXfer
CyU3PDmaChannelReset

Parameters

handle	Handle to the channel to be aborted.

5.14.4.2 CyU3PReturnStatus_t CyU3PDmaChannelCacheControl (CyU3PDmaChannel * *handle*, CyBool_t *isDmaHandleDCache*)

This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.

Description

The DMA driver in the FX3 library assumes that any manual data transfer on a DMA channel can be affected by the Data cache. Therefore, it ensures that the data buffer region is evicted from the cache before reading any data from a newly produced buffer. It also ensures that the data buffer region is written back to memory before commiting the buffer to the consumer.

These cache operations can limit the transfer performance obtained on the DMA channel in some cases. If the caller can ensure that the contents of the data buffer are accessed by the firmware only in very rare cases, it is possible to get better performance by removing these cache operations.

This API is used to selectively turn off the data cache handling on a per DMA channel basis. This should only be used with caution, and the caller should ensure that the cache APIs are used directly where required.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not idle (configured state).

See Also

CyU3PDmaChannel
CyU3PDeviceCacheControl
CyU3PDmaChannelCreate

Parameters

handle	Handle to the DMA channel.
isDmaHandleD-Cache	Whether to enable handling or not.

5.14.4.3 CyU3PReturnStatus_t CyU3PDmaChannelCommitBuffer (CyU3PDmaChannel * *handle*, uint16_t *count*, uint16_t *bufStatus*)

Commit a data buffer to be sent to the consumer.

Description

This function is generally used with manual DMA channels, and allows the user to commit a data buffer which is to be sent out to the consumer. This operation is not valid for channels of type MANUAL_IN.

The count provided is the exact size of the data that is to be sent out of the consumer socket.

This API can be used with AUTO DMA channels in the special case where the consumer socket is suspended. This allows the user to modify the data content of a partial data buffer, and then commit it for sending out of the device.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the count is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if this sequence is not permitted.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannelGetBuffer
CyU3PDmaChannelDiscardBuffer

Parameters

handle	Handle to the DMA channel to be modified.
count	Size of data in the buffer being committed. The buffer address is implicit and is fetched from the active descriptor for the channel.
bufStatus	Current status (end of transfer bit) of the buffer being committed. The occupied bit will automatically be set by the API.

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

Create a one-to-one socket DMA channel.

Description

Create a normal (one-to-one socket) DMA channel using the configuration parameters specified. The DMA channel structure is expected to be allocated by the caller, and a pointer to it should be passed in as a parameter. This function should not be called from a DMA callback function.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_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.

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

Parameters

handle	Pointer to channel structure that should be initialized.
type	Type of DMA channel desired.
config	Channel configuration parameters.

5.14.4.5 CyU3PReturnStatus_t CyU3PDmaChannelDestroy (CyU3PDmaChannel * *handle*)

Destroy a one-to-one socket DMA channel.

Description

This function frees up a one-to-one socket DMA channel once it is no longer required. This should not be called from a DMA callback.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel structure is not valid.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelCreate

Parameters

handle	Pointer to DMA channel structure to be de-initialized.

5.14.4.6 CyU3PReturnStatus_t CyU3PDmaChannelDiscardBuffer (CyU3PDmaChannel * *handle*)

Drop a data buffer without sending out to the consumer.

Description

This function drops the content of the active buffer by moving the consumer socket ahead to the next buffer. This function is generally used with DMA channels of type MANUAL and MANUAL_IN.

In the case of AUTO DMA channels, this API is used in the special case where the consumer socket is suspended using the CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF option. This allows the user to drop the content of a partially filled buffer without sending it out to the consumer.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if this sequence is not permitted.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelGetBuffer
CyU3PDmaChannelCommitBuffer

Parameters

handle	Handle to the DMA channel to be modified.

5.14.4.7 CyU3PReturnStatus_t CyU3PDmaChannelGetBuffer (CyU3PDmaChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint32_t *waitOption*)

Get the current buffer pointer.

Description

This function waits until a buffer is ready on the channel, and then returns a pointer to the buffer status.

The ready condition is defined differently for different DMA channel types:

1. For MANUAL and MANUAL_IN DMA channels, a ready buffer is one which has been filled with data by the producer.
2. For MANUAL_OUT channels, a ready buffer is an empty buffer that can be filled by the firmware.
3. For AUTO channels, this API can only be called when the consumer socket is suspended. This mechanism is supported to facilitate reading the buffer status when the channel has been suspended using the CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF option.

If this function is called from a DMA callback, the wait option should be set to CYU3P_NO_WAIT.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer parameters are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if this sequence is not permitted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaBuffer_t
CyU3PDmaChannel
CyU3PDmaChannelSetXfer
CyU3PDmaChannelCommitBuffer
CyU3PDmaChannelDiscardBuffer

Parameters

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

5.14.4.8 CyU3PDmaChannel* CyU3PDmaChannelGetHandle (CyU3PDmaSocketId_t *sckId*)

Fetch a handle to the DMA channel corresponding to the specified socket.

Description

This function is used to identify if there is any DMA channel associated with a socket.

Return value

Handle of the channel associated with the specified socket. A NULL return indicates that the socket has not been associated with any DMA channel.

See Also

CyU3PDmaChannel
CyU3PDmaSocketId_t
CyU3PDmaChannelCreate
CyU3PDmaChannelDestroy

Parameters

sckId	ID of the socket whose channel handle is required.

5.14.4.9 CyU3PReturnStatus_t CyU3PDmaChannelGetStatus (CyU3PDmaChannel * *handle*, CyU3PDmaState_t * *state*, uint32_t * *prodXferCount*, uint32_t * *consXferCount*)

This function returns the current channel status.

Description

This function returns the current state of the DMA channel as well as the current transfer counts on both producer and consumer sockets.

Note

The FX3 device only updates the transfer count values at buffer boundaries, and it is not possible to identify the transfer count at a partial buffer level.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaState_t
CyU3PDmaChannelSetXfer

Parameters

handle	Handle to the DMA channel to query.
state	Output parameter that will be filled with state of the channel.
prodXferCount	Output parameter that will be filled with transfer count on the producer socket in DMA mode units. Will return zero in the case of MANUAL_OUT channels.
consXferCount	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units. Will return zero in the case of MANUAL_IN channels.

5.14.4.10 CyU3PReturnStatus_t CyU3PDmaChannelReset (CyU3PDmaChannel * *handle*)

Aborts and resets a DMA channel.

Description

The function aborts the ongoing transfer on a DMA channel, and restores all sockets and descriptors to their initial state. At the end of the Reset call, all resources associated with the channel are back in the state that they are in immediately after channel creation.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelCreate
CyU3PDmaChannelAbort

Parameters

handle	Handle to the channel to be reset.

5.14.4.11 CyU3PReturnStatus_t CyU3PDmaChannelResume (CyU3PDmaChannel * *handle*, CyBool_t *isProdResume*, CyBool_t *isConsResume*)

Resume a suspended DMA channel.

Description

The function can be called to resume a suspended DMA channel. It can only be called when the channel was suspended from an active state. The producer and consumer suspend conditions can be individually cleared.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the channel type is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel was not started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannelSetSuspend

Parameters

handle	Handle to the channel to be resumed.
isProdResume	Whether to resume the producer socket.
isConsResume	Whether to resume the consumer socket.

5.14.4.12 CyU3PReturnStatus_t CyU3PDmaChannelSetSuspend (CyU3PDmaChannel * *handle*, CyU3PDmaSckSuspType_t *prodSusp*, CyU3PDmaSckSuspType_t *consSusp*)

Set the suspend options for the sockets associated with a DMA channel.

Description

The function sets the suspend options for the sockets. The sockets are by default set to SUSP_NONE option. The API can be called only when the channel is in configured state or in active state. The suspend options are applied only in the active state (SetXfer mode) and is a don't care in the override mode of operation.

For manual channels, suspending the channel is largely not required as each buffer needs to be manually committed. However, these options are still available for manual DMA channels.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the channel type/suspend options are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelResume
CyU3PDmaChannelReset

Parameters

handle	Handle to the channel to be modified.
prodSusp	Suspend option for the producer socket.
consSusp	Suspend option for the consumer socket.

5.14.4.13 CyU3PReturnStatus_t CyU3PDmaChannelSetupRecvBuffer (CyU3PDmaChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*)

Receive data into a user provided DMA buffer.

Description

This function initiates a read of incoming data from a DMA producer into a user provided buffer. This operation is performed in override mode of the DMA channel, and can only be initiated when the channel is in configured state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer size should be a multiple of 16 bytes.
2. If the data cache is enabled then, the buffer should be 32 byte aligned and a multiple of 32 bytes. This is to match the 32 byte cache line. 32 byte check is not enforced by the API as the buffer can be over-allocated.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer parameters are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_ALREADY_STARTED - if the DMA channel is already started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaBuffer_t
CyU3PDmaChannel
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelWaitForRecvBuffer

Parameters

handle	Handle to the DMA channel to be modified.
buffer_p	Pointer to structure containing the address and size of the buffer to be filled up with received data.

5.14.4.14 CyU3PReturnStatus_t CyU3PDmaChannelSetupSendBuffer (CyU3PDmaChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*)

Send the contents of a user provided buffer to the consumer.

Description

This function initiates the sending of the content of a user provided buffer to the consumer of a DMA channel. This function is an override on the normal behavior of the DMA channel and can only be called when the channel is in the configured state.

The buffers used for DMA operations are expected to be allocated using the CyU3PDmaBufferAlloc call. If this is not the case, then the buffer has to be over allocated in such a way that the full buffer should be 32 byte aligned and should be a multiple of 32 bytes in size.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer parameters are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_ALREADY_STARTED - if the DMA channel is already started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaBuffer_t
CyU3PDmaChannelCreate
CyU3PDmaChannelReset
CyU3PDmaChannelSetupRecvBuffer

Parameters

handle	Handle to the DMA channel to be modified.
buffer_p	Pointer to structure containing address, size and status of the DMA buffer to be sent out.

5.14.4.15 CyU3PReturnStatus_t CyU3PDmaChannelSetWrapUp (CyU3PDmaChannel * *handle*)

Wraps up the current active buffer for the channel from the producer side.

Description

Data received by a DMA producer socket is only committed (made available to the consumer) when the buffer is completely filled up, or when the producer signals and End Of Packet (EOP) condition. There may be cases where these conditions are not met, and a partially filled buffer is blocked waiting for more data.

This function can be used to forcibly commit the contents of the active DMA buffer on the producer side of the channel. A DMA produce event will be generated as a result of this API call, and the producer socket will move on to the next buffer in the chain without getting suspended.

The function cannot be used with MANUAL_OUT channels, and can only be called when the channel is the active or consumer override state.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaState_t
CyU3PDmaChannelGetStatus
CyU3PDmaChannelSetXfer
CyU3PDmaChannelGetBuffer
CyU3PDmaChannelCommitBuffer

Parameters

handle	Handle to the channel to be modified.

5.14.4.16 CyU3PReturnStatus_t CyU3PDmaChannelSetXfer (CyU3PDmaChannel * *handle*, uint32_t *count*)

Setup a one-to-one DMA channel for data transfer.

Description

The sockets corresponding to a DMA channel are left disabled when the channel is created, so that transfers are started only when desired by the user. This function enables a DMA channel to transfer a specified amount of data before suspending again. An infinite transfer can be started by specifying a transfer size of 0.

This function should be called only when the channel is in the CY_U3P_DMA_CONFIGURED state.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if no buffers were allocated for this DMA channel.

CY_U3P_ERROR_ALREADY_STARTED - if the channel is already active.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannel
CyU3PDmaChannelCreate
CyU3PDmaChannelGetStatus
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForCompletion
CyU3PDmaChannelReset

Parameters

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

5.14.4.17 CyU3PReturnStatus_t CyU3PDmaChannelUpdateMode (CyU3PDmaChannel * *handle*, CyU3PDmaMode_t *dmaMode*)

Update the DMA mode for a one-to-one DMA channel.

Description

The DMA mode of a channel decides whether transfers on the channel are tracked in terms of bytes or buffers. The mode is normally specified at the time of channel creation and left unmodified there-after. This API can be used if the mode for an existing channel needs to be changed. This can only be called when the channel is in the configured (just created or reset) state.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if DMA mode is invalid.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the Configured state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMode_t
CyU3PDmaChannel
CyU3PDmaChannelCreate
CyU3PDmaChannelGetStatus

Parameters

handle	Handle to DMA channel to be modified.
dmaMode	Desired DMA operating mode. Can be byte mode or buffer mode.

5.14.4.18 CyU3PReturnStatus_t CyU3PDmaChannelWaitForCompletion (CyU3PDmaChannel * *handle*, uint32_t *waitOption*)

Wait for the current DMA transaction to complete.

Description

This function waits until the current transfer on the DMA channel is completed, or until the specified timeout period has elapsed. This function is not supported if the DMA channel has been configured for infinite transfers.

This function should not be called from a DMA callback function. If the function returns CY_U3P_ERROR_TIMEOUT, the wait API call can be repeated without affecting the data transfer.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannelSetXfer
CyU3PDmaChannelSetupSendBuffer
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForRecvBuffer

Parameters

handle	Handle to the DMA channel to wait on.
waitOption	Duration for which to wait.

5.14.4.19 CyU3PReturnStatus_t CyU3PDmaChannelWaitForRecvBuffer (CyU3PDmaChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint32_t *waitOption*)

Wait until an override read operation is completed.

Description

This function waits until the read operation initiated through the CyU3PDmaChannelSetupRecvBuffer API is completed. Once the transfer is completed, it returns the status of the buffer that was filled with data. If not, a timeout error is returned. It is possible to retry this wait API without resetting or aborting the channel.

This function must not be called from a DMA callback function.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer parameters are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_INVALID_SEQUENCE - if this sequence is not permitted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

See Also

CyU3PDmaChannel
CyU3PDmaChannelSetupRecvBuffer
CyU3PDmaChannelWaitForCompletion

Parameters

handle	Handle to the DMA channel on which to wait.
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.
waitOption	Duration to wait for the receive completion.

5.14.4.20 void CyU3PDmaEnableMulticast (void)

Enable creation and management of DMA multicast channels.

Description

It is expected that DMA multicast channels will only rarely be used in FX3 applications. Since the multichannel creation code takes in the channel type as a parameter and then calls the appropriate handler functions, the code to setup and work with multicast channels gets linked into any FX3 application that uses multichannels, leading to un-necessary loss of code space. This function is provided to prevent this memory loss.

The multicast channel code will only be linked into the FX3 application if this function has been called. Therefore, this function needs to be called by the application before any multicast channels are created.

Return value

None

See Also

CyU3PDmaMultiChannelCreate

5.14.4.21 CyU3PReturnStatus_t CyU3PDmaMulticastSocketSelect (CyU3PDmaMultiChannel * *chHandle*, uint32_t *consMask*)

Select the active consumers on a multicast DMA channel.

Description

A multicast DMA channel has two or more consumer sockets, all of which receive data sent obtained through the producer socket. There may be cases where the firmware application needs to dynamically change the list of active consumers (that will receive the data). This API is used to select the active consumers on a multicast DMA channel.

The consMask parameter is a bitmask which represents the active consumers. Bit n of this bitmask needs to be set if consumer n is to be activated. By default, all consumers on the channel are left active. This API can only be used when the DMA channel is the idle (configured) state.

Return value

CY_U3P_SUCCESS if the socket selection is successful. CY_U3P_ERROR_NOT_SUPPORTED if the channel specified is not a multicast channel. CY_U3P_ERROR_ALREADY_STARTED if the channel is already active (SetXfer called).

5.14.4.22 CyU3PReturnStatus_t CyU3PDmaMultiChannelAbort (CyU3PDmaMultiChannel * *handle*)

Abort the ongoing transfer on a DMA multi-channel.

Description

The function shall abort all the producer and consumer sockets associated with the channel. The data in transition is lost and any active transaction cannot be resumed. This function leaves the channel in an aborted state and requires a reset before the channel can be used again.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelReset
CyU3PDmaMultiChannelGetStatus

Parameters

handle	Handle to the multi-channel to be aborted.

5.14.4.23 CyU3PReturnStatus_t CyU3PDmaMultiChannelCacheControl (CyU3PDmaMultiChannel * *handle*, CyBool_t *isDmaHandleDCache*)

This function is used to enable/disable the Data cache coherency handling on a per DMA channel basis.

Description

The DMA driver in the FX3 library assumes that any manual data transfer on a DMA channel can be affected by the Data cache. Therefore, it ensures that the data buffer region is evicted from the cache before reading any data from a newly produced buffer. It also ensures that the data buffer region is written back to memory before commiting the buffer to the consumer.

These cache operations can limit the transfer performance obtained on the DMA channel in some cases. If the caller can ensure that the contents of the data buffer are accessed by the firmware only in very rare cases, it is possible to get better performance by removing these cache operations.

This API is used to selectively turn off the data cache handling on a per DMA channel basis. This should only be used with caution, and the caller should ensure that the cache APIs are used directly where required.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not idle (configured state).

See Also

CyU3PDmaMultiChannel CyU3PDeviceCacheControl

Parameters

handle	Handle to the DMA channel.
isDmaHandleDCache	Whether to enable handling or not.

5.14.4.24 CyU3PReturnStatus_t CyU3PDmaMultiChannelCommitBuffer (CyU3PDmaMultiChannel * *handle*, uint16_t *count*, uint16_t *bufStatus*)

Commit the buffer to be sent to the consumer.

Description

This function is generally used with manual DMA channels, and allows the user to commit a data buffer which is to be sent out to the consumer. In the case of one-to-many channels, the buffer will be sent out through the consumer channels on a round-robin basis.

The count provided is the exact size of the data that is to be sent out of the consumer socket.

This API can be used with many-to-one AUTO DMA channels in the special case where the consumer socket is suspended. This allows the user to modify the data content of a partial data buffer, and then commit it for sending out of the device.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelGetBuffer
CyU3PDmaMultiChannelDiscardBuffer

Parameters

handle	Handle to the multi-channel to be modified.
count	Size of the memory buffer being committed. The address of the buffer is implicit and is taken from the active descriptor.
bufStatus	Status of the buffer being committed.

5.14.4.25 CyU3PReturnStatus_t CyU3PDmaMultiChannelCreate (CyU3PDmaMultiChannel * *handle*, CyU3PDmaMultiType_t *type*, CyU3PDmaMultiChannelConfig_t * *config*)

Create a multi-socket DMA channel with the specified parameters.

Description

This function is used to create a multi-socket DMA channel based on the specified parameters. The multi-channel structure is to be allocated by the caller, and a pointer to this is to be passed as parameter to this function.

This function must not be called from a DMA callback function.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if any of the configuration parameters are invalid.

CY_U3P_ERROR_MEMORY_ERROR - if the memory required for the channel could not be allocated.

CY_U3P_ERROR_INVALID_SEQUENCE - if a multicast channel is being created without calling CyU3PDmaEnableMulticast.

See Also

CyU3PDmaMultiType_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelConfig_t
CyU3PDmaMultiChannelDestroy
CyU3PDmaChannelCreate

Parameters

handle	Pointer to multi-channel structure that is to be initialized.
type	Type of DMA channel to be created.
config	Configuration information about the channel to be created.

5.14.4.26 CyU3PReturnStatus_t CyU3PDmaMultiChannelDestroy (CyU3PDmaMultiChannel * *handle*)

Destroy a multi-socket DMA channel.

Description

This function is used to free up the resources used by a DMA multi-channel when it is no longer required. This function should not be called from a DMA callback function.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelCreate
CyU3PDmaChannelDestroy

5.14.4.27 CyU3PReturnStatus_t CyU3PDmaMultiChannelDiscardBuffer (CyU3PDmaMultiChannel * *handle*)

Drop a data buffer without sending out to the consumer.

Description

This function drops the content of the active buffer by moving the consumer socket ahead to the next buffer. This function is generally used with DMA channels of type MANUAL and MANUAL_IN.

In the case of many-to-one AUTO DMA channels, this API is used in the special case where the consumer socket is suspended using the CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF option. This allows the user to drop the content of a partially filled buffer without sending it out to the consumer.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelGetBuffer
CyU3PDmaMultiChannelCommitBuffer

Parameters

handle	Handle to the multi-channel to be modified.

5.14.4.28 CyU3PReturnStatus_t CyU3PDmaMultiChannelGetBuffer (CyU3PDmaMultiChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint32_t *waitOption*)

Get the current buffer pointer.

Description

This function waits until a buffer is ready on the channel, and then returns a pointer to the buffer status.

In the case of many-to-one DMA channels, the buffers are received from the producer sockets in a round-robin fashion. The first producer is selected through the multiSckOffset parameter passed to the CyU3PDmaMultiChannelSetXfer API.

The ready condition is defined differently for different DMA channel types:

1. For MANUAL and MANUAL_IN DMA channels, a ready buffer is one which has been filled with data by the producer.
2. For MANUAL_OUT channels, a ready buffer is an empty buffer that can be filled by the firmware.
3. For many-to-one AUTO channels, this API can only be called when the consumer socket is suspended. This mechanism is supported to facilitate reading the buffer status when the channel has been suspended using the CY_U3P_DMA_SCK_SUSP_CONS_PARTIAL_BUF option.

If this function is called from a DMA callback, the wait option should be set to CYU3P_NO_WAIT.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaBuffer_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelDestroy
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelCommitBuffer
CyU3PDmaMultiChannelDiscardBuffer

Parameters

handle	Handle to the multi-channel to be modified.
buffer_p	Output parameter that will be filled with address, size and status of the buffer with received data.
waitOption	Duration for which to wait for data.

5.14.4.29 CyU3PDmaMultiChannel* CyU3PDmaMultiChannelGetHandle (CyU3PDmaSocketId_t *sckId*)

Identifies the multi-channel that is associated with the specified socket.

Description

This function returns a pointer to the multi-channel that is associated with the specified DMA socket. This function will return NULL if there are no channels associated with the specified socket.

Return value

Handle to the Multi-channel structure corresponding to the socket.

See Also

CyU3PDmaSocketId_t
CyU3PDmaMultiChannel

Parameters

sckId	ID of the socket whose channel handle is required.

5.14.4.30 CyU3PReturnStatus_t CyU3PDmaMultiChannelGetStatus (CyU3PDmaMultiChannel * *handle*, CyU3PDmaState_t * *state*, uint32_t * *prodXferCount*, uint32_t * *consXferCount*, uint8_t *sckIndex*)

This functions returns the current multi-channel status.

Description

This function returns the current state of the DMA channel as well as the current transfer counts on both producer and consumer sockets.

Note

The FX3 device only updates the transfer count values at buffer boundaries, and it is not possible to identify the transfer count at a partial buffer level.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the sckIndex is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaState_t
CyU3PDmaMultiChannelSetXfer

Parameters

handle	Handle to the DMA channel to query.
state	Output parameter that will be filled with state of the channel.
prodXferCount	Output parameter that will be filled with transfer count on the producer socket in DMA mode units.
consXferCount	Output parameter that will be filled with transfer count on the consumer socket in DMA mode units.
sckIndex	The socket index for retrieving information transfer counts.

5.14.4.31 CyU3PReturnStatus_t CyU3PDmaMultiChannelReset (CyU3PDmaMultiChannel * *handle*)

Abort ongoing transfers on and resets a DMA multi-channel.

Description

The function aborts the ongoing transfer on a DMA channel, and restores all sockets and descriptors to their initial state. At the end of the Reset call, all resources associated with the channel are back in the state that they are in immediately after channel creation.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelAbort
CyU3PDmaMultiChannelGetStatus

Parameters

handle	Handle to the multi-channel to be reset.

5.14.4.32 CyU3PReturnStatus_t CyU3PDmaMultiChannelResume (CyU3PDmaMultiChannel * *handle*, CyBool_t *isProdResume*, CyBool_t *isConsResume*)

Resume a suspended DMA multi-channel.

Description

The function can be called to resume a suspended DMA channel. It can only be called when the channel was suspended from an active state.

For many-to-one channels, only the consumer side can be resumed; and for one-to-many channels, only the producer side can be resumed.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the resume options are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel CyU3PDmaChannelResume CyU3PDmaMultiChannelSetSuspend

Parameters

handle	Handle to the multi-channel to be resumed.
isProdResume	Whether to resume the producer socket.
isConsResume	Whether to resume the consumer socket.

5.14.4.33 CyU3PReturnStatus_t CyU3PDmaMultiChannelSetSuspend (CyU3PDmaMultiChannel * *handle*, CyU3PDmaSckSuspType_t *prodSusp*, CyU3PDmaSckSuspType_t *consSusp*)

Update the suspend options for the sockets associated with a DMA multi-channel.

Description

The function sets the suspend options for the sockets. The sockets are by default set to SUSP_NONE option. The API can be called only when the channel is in configured state or in active state. The suspend options are applied only in the active state (SetXfer mode) and is a don't care in the override mode of operation.

For manual channels, suspending the channel is largely not required as each buffer needs to be manually committed. However, these options are still available for manual DMA channels.

Suspend options can only be set for the end where there is a single socket. i.e., Suspend options can only be set for the producer side on one-to-many and multicast channels; and only for the consumer side on many-to-one channels.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the suspend options are invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaSckSuspType_t
CyU3PDmaChannelSetSuspend
CyU3PDmaMultiChannelResume

Parameters

handle	Handle to the multi-channel to be suspended.
prodSusp	Suspend option for the producer socket.
consSusp	Suspend option for the consumer socket.

5.14.4.34 CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupRecvBuffer (CyU3PDmaMultiChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint16_t *multiSckOffset*)

Receive data into a user provided DMA buffer.

Description

This function initiates a read of incoming data from a DMA producer into a user provided buffer. This operation is performed in override mode of the DMA channel, and can only be initiated when the channel is in configured state.

The buffer that is passed as parameter for receiving the data has the following restrictions:

1. The buffer size should be a multiple of 16 bytes.
2. If the data cache is enabled then, the buffer should be 32 byte aligned and a multiple of 32 bytes. This is to match the 32 byte cache line. 32 byte check is not enforced by the API as the buffer can be over-allocated.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if an invalid parameters are passed.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_ALREADY_STARTED - if the DMA channel was already started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaBuffer_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelWaitForRecvBuffer

Parameters

handle	Handle to the DMA multi channel to be modified.
buffer_p	Pointer to structure containing the address and size of the buffer to be filled up with received data.
multiSckOffset	Producer Socket id to receive the data. For a one to many channel, this should be zero; and for a many to one channel, this would be a producer socket offset.

5.14.4.35 CyU3PReturnStatus_t CyU3PDmaMultiChannelSetupSendBuffer (CyU3PDmaMultiChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint16_t *multiSckOffset*)

Send the contents of a user provided buffer to the consumer.

Description

This function initiates the sending of the content of a user provided buffer to the consumer of a DMA channel. This function is an override on the normal behavior of the DMA channel and can only be called when the channel is in the configured state.

The buffers used for DMA operations are expected to be allocated using the CyU3PDmaBufferAlloc call. If this is not the case, then the buffer has to be over allocated in such a way that the full buffer should be 32 byte aligned and should be a multiple of 32 bytes in size.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if an invalid parameters are passed.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_ALREADY_STARTED - if the DMA channel was already started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaBuffer_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetupRecvBuffer

Parameters

handle	Handle to the DMA multi channel to be modified.
buffer_p	Pointer to structure containing address, size and status of the DMA buffer to be sent out.
multiSckOffset	Consumer Socket id to send the data. For a many to one channel, this should be zero; and for a one to many channel, this would be a consumer socket offset.

5.14.4.36 CyU3PReturnStatus_t CyU3PDmaMultiChannelSetWrapUp (CyU3PDmaMultiChannel * *handle*, uint16_t *multiSckOffset*)

Wraps up the current active buffer on the producer socket.

Description

This API is used to forcibly commit a DMA buffer to the consumer, and is useful in the case where data transfer has abruptly stopped without the producer being able to commit the data buffer.

The function can be called only when the channel is in active mode or in consumer override mode.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the multiSckOffset is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaChannelSetWrapUp
CyU3PDmaMultiChannel

Parameters

handle	Handle to the channel to be modified.
multiSckOffset	Socket id to wrapup. For a many to one channel, this would be a producer socket offset; and for a one to many channel, this would always be zero.

5.14.4.37 CyU3PReturnStatus_t CyU3PDmaMultiChannelSetXfer (CyU3PDmaMultiChannel * *handle*, uint32_t *count*, uint16_t *multiSckOffset*)

Prepare a DMA multi-channel for data transfer.

Description

This function prepares the sockets involved in a DMA multi-channel for data transfer. This function can only be called when the channel is in the CY_U3P_DMA_CONFIGURED state. The amount of data to be transferred before the channel gets suspended is specified as a parameter, and can be set to 0 to start infinite transfers.

The multiSckOffset parameter specifies the index of the producer (in the case of many-to-one channels) or the consumer (one-to-many channels) that will transfer the first buffer. The rest of the sockets will be used in the sequence specified at channel creation.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the multiSckOffset is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_ALREADY_STARTED - if the DMA channel was already started.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelCreate
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForCompletion
CyU3PDmaMultiChannelReset
CyU3PDmaMultiChannelGetStatus

Parameters

handle	Handle to the multi-channel to be modified.
count	Size of the transfer to be started. Can be zero to denote infinite data transfer.
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.

5.14.4.38 CyU3PReturnStatus_t CyU3PDmaMultiChannelUpdateMode (CyU3PDmaMultiChannel * *handle*, CyU3PDmaMode_t *dmaMode*)

Update the DMA mode for a multi-channel.

Description

The DMA mode for a channel specifies the units in which data transfers on the channel are setup and measured. This is generally set during channel creation and retained unchanged there-after. This function can be used to dynamically change the DMA mode for a multi-channel.

The function can only be called when the DMA channel is in the configured state.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - if the DMA mode is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the Configured state.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMode_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelCreate
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelGetStatus

Parameters

handle	Handle to the multi-channel to be modified.
dmaMode	Desired DMA mode.

5.14.4.39 CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForCompletion (CyU3PDmaMultiChannel * *handle*, uint32_t *waitOption*)

Wait for the current DMA transaction to complete on a DMA multi-channel.

Description

This function waits until the current transfer on the DMA channel is completed, or until the specified timeout period has elapsed. This function is not supported if the DMA channel has been configured for infinite transfers.

This function should not be called from a DMA callback function. If the function returns CY_U3P_ERROR_TIMEOUT, the wait API call can be repeated without affecting the data transfer.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

See Also

CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetXfer
CyU3PDmaMultiChannelSetupSendBuffer
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForRecvBuffer
CyU3PDmaMultiChannelReset

Parameters

handle	Handle to the multi-channel to wait on.
waitOption	Duration for which to wait.

5.14.4.40 CyU3PReturnStatus_t CyU3PDmaMultiChannelWaitForRecvBuffer (CyU3PDmaMultiChannel * *handle*, CyU3PDmaBuffer_t * *buffer_p*, uint32_t *waitOption*)

Wait until an override read operation is completed.

Description

This function waits until the read operation initiated through the CyU3PDmaMultiChannelSetupRecvBuffer API is completed. Once the transfer is completed, it returns the status of the buffer that was filled with data. If not, a timeout error is returned. It is possible to retry this wait API without resetting or aborting the channel.

This function must not be called from a DMA callback function.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_NULL_POINTER - if any pointer passed as parameter is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - if the DMA channel was not configured.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is not supported for the DMA channel type.

CY_U3P_ERROR_INVALID_SEQUENCE - if the DMA channel is not in the required state.

CY_U3P_ERROR_DMA_FAILURE - if the DMA transfer failed.

CY_U3P_ERROR_ABORTED - if the DMA transfer was aborted.

CY_U3P_ERROR_TIMEOUT - if the DMA transfer timed out.

CY_U3P_ERROR_MUTEX_FAILURE - if the DMA channel mutex could not be acquired.

CY_U3P_ERROR_NOT_STARTED - if the DMA channel is not started.

See Also

CyU3PDmaBuffer_t
CyU3PDmaMultiChannel
CyU3PDmaMultiChannelSetupRecvBuffer
CyU3PDmaMultiChannelWaitForCompletion

Parameters

handle	Handle to the DMA channel on which to wait.
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.
waitOption	Duration to wait for the receive completion.

5.15 firmware/u3p_firmware/inc/cyu3error.h File Reference

This file lists the error codes that are returned by the firmware library functions.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Typedefs

- typedef enum CyU3PErrorCode_t CyU3PErrorCode_t

  List of error codes defined and returned by the FX3 firmware library.

Enumerations

- enum CyU3PErrorCode_t {
  CY_U3P_SUCCESS = 0, CY_U3P_ERROR_DELETED, CY_U3P_ERROR_BAD_POOL, CY_U3P_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_P-

OINTER, 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_STANDBY_FAILED,
CY_U3P_ERROR_INVALID_VOLTAGE_RANGE = 0x60, CY_U3P_ERROR_CARD_WRONG_RESPONSE, CY_U3P_ERROR_UNSUPPORTED_CARD, CY_U3P_ERROR_CARD_WRONG_STATE,
CY_U3P_ERROR_CMD_NOT_SUPPORTED, CY_U3P_ERROR_CRC, CY_U3P_ERROR_INVALID_ADDR, CY_U3P_ERROR_INVALID_UNIT,
CY_U3P_ERROR_INVALID_DEV, CY_U3P_ERROR_ALREADY_PARTITIONED, CY_U3P_ERROR_NOT_PARTITIONED, CY_U3P_ERROR_BAD_PARTITION,
CY_U3P_ERROR_READ_WRITE_ABORTED, CY_U3P_ERROR_WRITE_PROTECTED, CY_U3P_ERROR_SIB_INIT, CY_U3P_ERROR_CARD_LOCKED,
CY_U3P_ERROR_CARD_LOCK_FAILURE, CY_U3P_ERROR_CARD_FORCE_ERASE, CY_U3P_ERROR_INVALID_BLOCKSIZE, CY_U3P_ERROR_INVALID_FUNCTION,
CY_U3P_ERROR_TUPLE_NOT_FOUND, CY_U3P_ERROR_IO_ABORTED, CY_U3P_ERROR_IO_SUSPENDED, CY_U3P_ERROR_ILLEGAL_CMD,
CY_U3P_ERROR_SDIO_UNKNOWN, CY_U3P_ERROR_BAD_CMD_ARG, CY_U3P_ERROR_UNINITIALIZED_FUNCTION, CY_U3P_ERROR_CARD_NOT_ACTIVE,
CY_U3P_ERROR_DEVICE_BUSY, CY_U3P_ERROR_NO_METADATA, CY_U3P_ERROR_CARD_UNHEALTHY, CY_U3P_ERROR_MEDIA_FAILURE,
CY_U3P_ERROR_NO_REENUM_REQUIRED = 0xFE, CY_U3P_ERROR_OPERN_DISABLED = 0xFF }

List of error codes defined and returned by the FX3 firmware library.

5.15.1 Detailed Description

This file lists the error codes that are returned by the firmware library functions.

5.15.2 Error Codes

The following return codes are used by the FX3 APIs to indicate success/failure and the cause of error in case of failure.

5.15.3 Typedef Documentation

5.15.3.1 typedef enum CyU3PErrorCode_t CyU3PErrorCode_t

List of error codes defined and returned by the FX3 firmware library.

Description

The error codes below help identify the cause of the failure. These errors could be returned by the base RTOS or by the FX3 API itself. The initial error codes are all defined and returned by the RTOS. The API specific error code start from CY_U3P_ERROR_BAD_ARGUMENT.

5.15.4 Enumeration Type Documentation

5.15.4.1 enum CyU3PErrorCode_t

List of error codes defined and returned by the FX3 firmware library.

Description

The error codes below help identify the cause of the failure. These errors could be returned by the base RTOS or by the FX3 API itself. The initial error codes are all defined and returned by the RTOS. The API specific error code start from CY_U3P_ERROR_BAD_ARGUMENT.

Enumerator

CY_U3P_SUCCESS Success code

CY_U3P_ERROR_DELETED The OS object being accessed has been deleted.

CY_U3P_ERROR_BAD_POOL Bad memory pool passed to a function.

CY_U3P_ERROR_BAD_POINTER Bad (NULL or unaligned) pointer passed to a function.

CY_U3P_ERROR_INVALID_WAIT Non-zero wait requested from interrupt context.

CY_U3P_ERROR_BAD_SIZE Invalid size value passed into a function.

CY_U3P_ERROR_BAD_EVENT_GRP Invalid event group passed into a function.

CY_U3P_ERROR_NO_EVENTS Failed to set/get the event flags specified.

CY_U3P_ERROR_BAD_OPTION Invalid task option value specified for the function.

CY_U3P_ERROR_BAD_QUEUE Invalid message queue passed to a function.

CY_U3P_ERROR_QUEUE_EMPTY The message queue being read is empty.

CY_U3P_ERROR_QUEUE_FULL The message queue being written to is full.

CY_U3P_ERROR_BAD_SEMAPHORE Invalid semaphore pointer passed to a function.

CY_U3P_ERROR_SEMGET_FAILED A semaphore get operation failed.

CY_U3P_ERROR_BAD_THREAD Invalid thread pointer passed to a function.

CY_U3P_ERROR_BAD_PRIORITY Invalid thread priority value passed to a function.

CY_U3P_ERROR_MEMORY_ERROR Failed to allocate memory.

CY_U3P_ERROR_DELETE_FAILED Failed to delete an object because it is not idle.

CY_U3P_ERROR_RESUME_FAILED Failed to resume a thread.

CY_U3P_ERROR_INVALID_CALLER OS function failed because the current caller is not allowed.

CY_U3P_ERROR_SUSPEND_FAILED Failed to suspend a thread.

CY_U3P_ERROR_BAD_TIMER Invalid timer pointer passed to a function.

CY_U3P_ERROR_BAD_TICK Invalid (0) tick value passed to a timer function.

CY_U3P_ERROR_ACTIVATE_FAILED Failed to activate a timer.

CY_U3P_ERROR_BAD_THRESHOLD Invalid thread pre-emption threshold value specified.

CY_U3P_ERROR_SUSPEND_LIFTED Thread suspension was cancelled.

CY_U3P_ERROR_WAIT_ABORTED Wait operation was aborted.

CY_U3P_ERROR_WAIT_ABORT_FAILED Failed to abort wait operation on a thread.

CY_U3P_ERROR_BAD_MUTEX Invalid Mutex pointer passed to a function.

CY_U3P_ERROR_MUTEX_FAILURE Failed to get a mutex.

CY_U3P_ERROR_MUTEX_PUT_FAILED Failed to put a mutex because it is not currently owned.

CY_U3P_ERROR_INHERIT_FAILED Error in priority inheritance.

CY_U3P_ERROR_NOT_IDLE Operation failed because relevant object is not idle or done.

CY_U3P_ERROR_BAD_ARGUMENT One or more parameters to a function are invalid.

CY_U3P_ERROR_NULL_POINTER A null pointer has been passed in unexpectedly.

CY_U3P_ERROR_NOT_STARTED The object/module being referred to has not been started.

CY_U3P_ERROR_ALREADY_STARTED An object/module that is already active is being started.

CY_U3P_ERROR_NOT_CONFIGURED Object/module referred to has not been configured.

CY_U3P_ERROR_TIMEOUT Timeout on relevant operation.

CY_U3P_ERROR_NOT_SUPPORTED Operation requested is not supported in current mode.

CY_U3P_ERROR_INVALID_SEQUENCE Invalid function call sequence.

CY_U3P_ERROR_ABORTED Function call failed as it was aborted by another thread/isr.

CY_U3P_ERROR_DMA_FAILURE DMA engine failed to completed requested operation.

CY_U3P_ERROR_FAILURE Failure due to a non-specific system error.

CY_U3P_ERROR_BAD_INDEX Bad index value was passed in as parameter. Ex: for string descriptor.

CY_U3P_ERROR_BAD_ENUM_METHOD Bad enumeration method specified.

CY_U3P_ERROR_INVALID_CONFIGURATION Invalid configuration specified.

CY_U3P_ERROR_CHANNEL_CREATE_FAILED Internal DMA channel creation failed.

CY_U3P_ERROR_CHANNEL_DESTROY_FAILED Internal DMA channel destroy failed.

CY_U3P_ERROR_BAD_DESCRIPTOR_TYPE Invalid descriptor type specified.

CY_U3P_ERROR_XFER_CANCELLED USB transfer was cancelled.

CY_U3P_ERROR_FEATURE_NOT_ENABLED When a USB feature like remote wakeup is not enabled.

CY_U3P_ERROR_STALLED When a USB request / data transfer is stalled.

CY_U3P_ERROR_BLOCK_FAILURE The block accessed has a fatal error and needs to be re-initialized.

CY_U3P_ERROR_LOST_ARBITRATION Loss of bus arbitration, invalid bus behaviour or bus busy.

CY_U3P_ERROR_STANDBY_FAILED Failed to enter standby mode because one or more wakeup events are active.

CY_U3P_ERROR_INVALID_VOLTAGE_RANGE Storage device's voltage range does not meet FX3S requirements.

CY_U3P_ERROR_CARD_WRONG_RESPONSE Incorrect response received from storage device.

CY_U3P_ERROR_UNSUPPORTED_CARD Storage device features are not supported by FX3S host.

CY_U3P_ERROR_CARD_WRONG_STATE Storage device failed to move to expected state.

CY_U3P_ERROR_CMD_NOT_SUPPORTED Storage device failed to support required commands.

CY_U3P_ERROR_CRC Response CRC error detected.

CY_U3P_ERROR_INVALID_ADDR Out of range address provided for read/write/erase access.

CY_U3P_ERROR_INVALID_UNIT Non-existent storage partition selected for transfer.

CY_U3P_ERROR_INVALID_DEV Access to port with no connected storage device.

CY_U3P_ERROR_ALREADY_PARTITIONED Request to partition a device which is already partitioned.

CY_U3P_ERROR_NOT_PARTITIONED Request to remove partitions on an unpartitioned device.

CY_U3P_ERROR_BAD_PARTITION Incorrect partition selected.

CY_U3P_ERROR_READ_WRITE_ABORTED Read/write transfer was aborted (user cancellation or timeout).

CY_U3P_ERROR_WRITE_PROTECTED Write request addressed to a write protected storage device.

CY_U3P_ERROR_SIB_INIT Storage driver initialization failed.

CY_U3P_ERROR_CARD_LOCKED Access to password locked SD card.

CY_U3P_ERROR_CARD_LOCK_FAILURE Failure while locking/unlocking the SD card.

CY_U3P_ERROR_CARD_FORCE_ERASE Failure during SD force erase operation.

CY_U3P_ERROR_INVALID_BLOCKSIZE Block size specified for SDIO device is not supported.

CY_U3P_ERROR_INVALID_FUNCTION Non-existent SDIO function being accessed.

CY_U3P_ERROR_TUPLE_NOT_FOUND Non-existent tuple of SDIO card being accessed.

CY_U3P_ERROR_IO_ABORTED IO operation to SDIO card aborted.

CY_U3P_ERROR_IO_SUSPENDED IO operation to SDIO card suspended.

CY_U3P_ERROR_ILLEGAL_CMD Invalid command sent to the SDIO card.

CY_U3P_ERROR_SDIO_UNKNOWN Generic error reported by SDIO card.

CY_U3P_ERROR_BAD_CMD_ARG SDIO command argument is out of range.

CY_U3P_ERROR_UNINITIALIZED_FUNCTION Access to uninitialized SDIO function.

CY_U3P_ERROR_CARD_NOT_ACTIVE Access to SDIO card which is not active.

CY_U3P_ERROR_DEVICE_BUSY The storage device is busy handling another request.

CY_U3P_ERROR_NO_METADATA No metadata present on card

CY_U3P_ERROR_CARD_UNHEALTHY Card RD/WR Threshold error crossed

CY_U3P_ERROR_MEDIA_FAILURE Card not responding to read/write transactions

CY_U3P_ERROR_NO_REENUM_REQUIRED FX3 booter supports the NoReEnumeration feature that enables to have a single USB enumeration across the FX3 booter and the final application. This error code is returned by CyU3PUsbStart () to indicate that the NoReEnumeration is successful and the sequence followed for the regular enumeration post CyU3PUsbStart () is to be skipped.

CY_U3P_ERROR_OPERN_DISABLED The requested feature/operation is not enabled in the current configuration.

5.16 firmware/u3p_firmware/inc/cyu3gpif.h File Reference

This file defines the GPIF related data structures and APIs in the FX3 SDK.

```
#include "cyu3types.h"
#include "cyu3dma.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PGpifWaveData

  Information on a single GPIF transition from one state to another.
- struct CyU3PGpifConfig_t

  Structure that holds all configuration inputs for the GPIF hardware.

Macros

- #define CYU3P_GPIF_NUM_STATES (256)

  Number of states supported by the GPIF hardware.
- #define CYU3P_GPIF_NUM_TRANS_FNS (32)

  Number of distinct transfer functions supported by the GPIF hardware.
- #define CYU3P_GPIF_INVALID_STATE (0xFFFF)

  Invalid state index for use in state machine control functions.
- #define CYU3P_PIB_THREAD_COUNT (4)

  Number of DMA threads on the GPIF (P-Port)
- #define CYU3P_PIB_SOCKET_COUNT (32)

  Number of DMA sockets on the GPIF (P-port)
- #define CYU3P_PIB_MAX_BURST_SETTING (14)

  Maximum burst size allowed for P-port sockets. The constant corresponds to Log(2) of size, which means that the max. size is 16 KB.

Typedefs

- typedef struct CyU3PGpifWaveData CyU3PGpifWaveData

  Information on a single GPIF transition from one state to another.
- typedef enum CyU3PGpifEventType CyU3PGpifEventType

  List of GPIF related events tracked by the driver.
- typedef enum CyU3PGpifComparatorType CyU3PGpifComparatorType

  List of supported comparators in the GPIF hardware.
- typedef void(∗ CyU3PGpifEventCb_t)(CyU3PGpifEventType event, uint8_t currentState)

  Callback type that is invoked to inform the application about GPIF events.
- typedef void(∗ CyU3PGpifSMIntrCb_t)(uint8_t stateId)

  Special callback type used for notification of GPIF state machine interrupts.
- typedef enum CyU3PGpifOutput_t CyU3PGpifOutput_t

  List of control outputs and flags from the GPIF hardware.
- typedef struct CyU3PGpifConfig_t CyU3PGpifConfig_t

  Structure that holds all configuration inputs for the GPIF hardware.

Enumerations

- enum CyU3PGpifEventType { CYU3P_GPIF_EVT_END_STATE = 0, CYU3P_GPIF_EVT_SM_INTERRUPT, CYU3P_GPIF_EVT_SWITCH_TIMEOUT, CYU3P_GPIF_EVT_ADDR_COUNTER, CYU3P_GPIF_EVT_DATA_COUNTER, CYU3P_GPIF_EVT_CTRL_COUNTER, CYU3P_GPIF_EVT_ADDR_COMP, CYU3P_GPIF_EVT_DATA_COMP, CYU3P_GPIF_EVT_CTRL_COMP, CYU3P_GPIF_EVT_CRC_ERROR }

  List of GPIF related events tracked by the driver.
- enum CyU3PGpifComparatorType { CYU3P_GPIF_COMP_CTRL = 0, CYU3P_GPIF_COMP_ADDR, CYU3P_GPIF_COMP_DATA }

  List of supported comparators in the GPIF hardware.
- enum CyU3PGpifOutput_t { CYU3P_GPIF_OP_ALPHA0 = 0, CYU3P_GPIF_OP_ALPHA1, CYU3P_GPIF_OP_ALPHA2, CYU3P_GPIF_OP_ALPHA3, CYU3P_GPIF_OP_BETA0 = 8, CYU3P_GPIF_OP_BETA1, CYU3P_GPIF_OP_BETA2, CYU3P_GPIF_OP_BETA3, CYU3P_GPIF_OP_THR0_READY = 16, CYU3P_GPIF_OP_THR1_READY, CYU3P_GPIF_OP_THR2_READY, CYU3P_GPIF_OP_THR3_READY, CYU3P_GPIF_OP_THR0_PART, CYU3P_GPIF_OP_THR1_PART, CYU3P_GPIF_OP_THR2_PART, CYU3P_GPIF_OP_THR3_PART, CYU3P_GPIF_OP_DMA_READY, CYU3P_GPIF_OP_PARTIAL, CYU3P_GPIF_OP_PPDRQ }

  List of control outputs and flags from the GPIF hardware.

Functions

- CyU3PReturnStatus_t CyU3PGpifLoad (const CyU3PGpifConfig_t ∗conf)

  Function to program the user defined state machine into the GPIF registers.
- void CyU3PGpifRegisterCallback (CyU3PGpifEventCb_t cbFunc)

  This function registers an event callback for the GPIF driver.
- void CyU3PGpifRegisterSMIntrCallback (CyU3PGpifSMIntrCb_t cb)

  Register a callback function for fast notification of GPIF state machine interrupts.
- CyU3PReturnStatus_t CyU3PGpifWaveformLoad (uint8_t firstState, uint16_t stateCnt, uint8_t ∗stateDataMap, CyU3PGpifWaveData ∗transitionData)

  Initialize the GPIF state machine table.

- CyU3PReturnStatus_t CyU3PGpifConfigure (uint8_t numRegs, const uint32_t *regData)
  Initialize the GPIF configuration registers.
- CyU3PReturnStatus_t CyU3PGpifRegisterConfig (uint16_t numRegs, uint32_t regData[][2])
  Initialize the GPIF configuration registers.
- CyU3PReturnStatus_t CyU3PGpifInitTransFunctions (uint16_t *fnTable)
  Initialize the GPIF transition function registers.
- void CyU3PGpifDisable (CyBool_t forceReload)
  Disables the GPIF state machine and hardware.
- CyU3PReturnStatus_t CyU3PGpifSMStart (uint8_t stateIndex, uint8_t initialAlpha)
  Start the waveform state machine from the specified state.
- CyU3PReturnStatus_t CyU3PGpifSMSwitch (uint16_t fromState, uint16_t toState, uint16_t endState, uint8_t initialAlpha, uint32_t switchTimeout)
  This function is used to start GPIF state machine execution from a desired state.
- void CyU3PGpifInitCtrlCounter (uint16_t initValue, uint16_t limit, CyBool_t reload, CyBool_t upCount, uint8_t outputBit)
  Function to configure the GPIF control counter.
- void CyU3PGpifInitAddrCounter (uint32_t initValue, uint32_t limit, CyBool_t reload, CyBool_t upCount, uint8_t increment)
  Function to configure the GPIF address counter.
- void CyU3PGpifInitDataCounter (uint32_t initValue, uint32_t limit, CyBool_t reload, CyBool_t upCount, uint8_t increment)
  Function to configure the GPIF data counter.
- CyU3PReturnStatus_t CyU3PGpifInitComparator (CyU3PGpifComparatorType type, uint32_t value, uint32_t mask)
  This function configures one of the comparators in the GPIF hardware.
- CyU3PReturnStatus_t CyU3PGpifSMControl (CyBool_t pause)
  Function to pause/resume the GPIF state machine.
- void CyU3PGpifControlSWInput (CyBool_t set)
  Function to set/clear the software controlled state machine input.
- CyU3PReturnStatus_t CyU3PGpifSocketConfigure (uint8_t threadIndex, CyU3PDmaSocketId_t socketNum, uint16_t watermark, CyBool_t flagOnData, uint8_t burst)
  Function to select an active socket on the P-port and to configure it.
- CyU3PReturnStatus_t CyU3PGpifReadDataWords (uint32_t threadIndex, CyBool_t selectThread, uint32_t numWords, uint32_t *buffer_p, uint32_t waitOption)
  Read a specified number of data words from the GPIF interface.
- CyU3PReturnStatus_t CyU3PGpifWriteDataWords (uint32_t threadIndex, CyBool_t selectThread, uint32_t numWords, uint32_t *buffer_p, uint32_t waitOption)
  Write a specified number of data words to the GPIF interface.
- CyU3PReturnStatus_t CyU3PGpifOutputConfigure (uint8_t ctrlPin, CyU3PGpifOutput_t opFlag, CyBool_t isActiveLow)
  Configure the functionality of a GPIF output pin.
- CyU3PReturnStatus_t CyU3PGpifGetSMState (uint8_t *curState_p)
  Get the current state of the GPIF state machine.

5.16.1 Detailed Description

This file defines the GPIF related data structures and APIs in the FX3 SDK. **Description**

The FX3 device includes a General Programmable Interface which can be used to connect it to any other processor, ASIC or FPGA using most master or slave protocols. The GPIF-II block on the FX3 device takes care of implementing the desired protocols by using a programmable state machine.

The GPIF-II APIs provide functions that can be used to configure the GPIF state machine to implement any desired protocol. Functions are also provided to control the execution of the state machine and to manage the actual data transfer across the GPIF configured interface.

5.16.2 Typedef Documentation

5.16.2.1 typedef enum CyU3PGpifComparatorType CyU3PGpifComparatorType

List of supported comparators in the GPIF hardware.

Description

This function lists the hardware comparators that are part of the GPIF block. Each of these comparators is configured by calling the CyU3PGpifInitComparator function with the corresponding type.

See Also

CyU3PGpifInitComparator

5.16.2.2 typedef struct CyU3PGpifConfig_t CyU3PGpifConfig_t

Structure that holds all configuration inputs for the GPIF hardware.

Description

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

See Also

CyU3PGpifLoad
CyU3PGpifWaveData

5.16.2.3 typedef void(* CyU3PGpifEventCb_t)(CyU3PGpifEventType event,uint8_t currentState)

Callback type that is invoked to inform the application about GPIF events.

Description

This type defines the signature for the callback function that is invoked by the GPIF driver to inform the user application about GPIF related events. A callback function of this type should be registered with the GPIF driver.

See Also

CyU3PGpifRegisterCallback

5.16.2.4 typedef enum CyU3PGpifEventType CyU3PGpifEventType

List of GPIF related events tracked by the driver.

Description

This type lists the various GPIF hardware and state machine related events that may be notified to the user application.

5.16.2.5 typedef enum CyU3PGpifOutput_t CyU3PGpifOutput_t

List of control outputs and flags from the GPIF hardware.

Description

The GPIF block in the FX3 device can generate a set of control outputs and DMA status flags that can be used by the external processor to control data transfers. This enumeration lists the various control outputs and flags that are supported by the GPIF.

See Also

CyU3PGpifOutputConfigure

5.16.2.6 typedef void(∗ CyU3PGpifSMIntrCb_t)(uint8_t stateId)

Special callback type used for notification of GPIF state machine interrupts.

Description

The generic GPIF interrupt callback mechanism provided by CyU3PGpifRegisterCallback delivers the callback in thread context. While this allows the user to call potentially blocking API calls from the callback, the latency from interrupt to callback will not be predictable and can extend to about 100 us.

This callback type provides notifications of only GPIF state machine interrupts (raised using the INTR_CPU action in the state machine) in interrupt context. This notification can be used to obtain a low latency notification of state machine interrupts. Please note that API calls that require a mutex get (such as DMA channel APIs) cannot be made directly from this callback.

See Also

CyU3PGpifRegisterSMIntrCallback

5.16.2.7 typedef struct CyU3PGpifWaveData CyU3PGpifWaveData

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

Description

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

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

See Also

CyU3PGpifWaveformLoad

5.16.3 Enumeration Type Documentation

5.16.3.1 enum CyU3PGpifComparatorType

List of supported comparators in the GPIF hardware.

Description

This function lists the hardware comparators that are part of the GPIF block. Each of these comparators is configured by calling the CyU3PGpifInitComparator function with the corresponding type.

See Also

CyU3PGpifInitComparator

Enumerator

CYU3P_GPIF_COMP_CTRL Control signals comparator.

CYU3P_GPIF_COMP_ADDR Address bus comparator.

CYU3P_GPIF_COMP_DATA Data bus comparator.

5.16.3.2 enum CyU3PGpifEventType

List of GPIF related events tracked by the driver.

Description

This type lists the various GPIF hardware and state machine related events that may be notified to the user application.

Enumerator

CYU3P_GPIF_EVT_END_STATE State machine has reached the designated end state.
CYU3P_GPIF_EVT_SM_INTERRUPT State machine has raised a software interrupt.
CYU3P_GPIF_EVT_SWITCH_TIMEOUT Desired state machine switch has timed out.
CYU3P_GPIF_EVT_ADDR_COUNTER Address counter has reached the limit.
CYU3P_GPIF_EVT_DATA_COUNTER Data counter has reached the limit.
CYU3P_GPIF_EVT_CTRL_COUNTER Control counter has reached the limit.
CYU3P_GPIF_EVT_ADDR_COMP Address comparator match has been obtained.
CYU3P_GPIF_EVT_DATA_COMP Data comparator match has been obtained.
CYU3P_GPIF_EVT_CTRL_COMP Control comparator match has been obtained.
CYU3P_GPIF_EVT_CRC_ERROR Incorrect CRC received on a read operation.

5.16.3.3 enum CyU3PGpifOutput_t

List of control outputs and flags from the GPIF hardware.

Description

The GPIF block in the FX3 device can generate a set of control outputs and DMA status flags that can be used by the external processor to control data transfers. This enumeration lists the various control outputs and flags that are supported by the GPIF.

See Also

CyU3PGpifOutputConfigure

Enumerator

CYU3P_GPIF_OP_ALPHA0 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 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 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.

5.16.4 Function Documentation

5.16.4.1 CyU3PReturnStatus_t CyU3PGpifConfigure (uint8_t *numRegs*, const uint32_t * *regData*)

Initialize the GPIF configuration registers.

Description

The GPIF hardware has a number of configuration registers that control the functioning of the state machine. These registers need to be initialized with values provided by the GPIF designer tool.

This function takes in an array containing the register values, and programs all of the configuration registers with these values.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_NOT_SUPPORTED - if a 32 bit GPIF configuration is being used on a part that does not support this.

CY_U3P_ERROR_BAD_ARGUMENT if any of the parameters are invalid.

CY_U3P_ERROR_ALREADY_STARTED if the GPIF hardware is running.

See Also

CyU3PGpifWaveformLoad
CyU3PGpifInitTransFunctions
CyU3PGpifRegisterConfig

Parameters

numRegs	Number of registers to configure (size of array).
regData	Pointer to uint32_t[] array, where element [i] contains the data to be loaded into GPIF config register "i".

5.16.4.2 void CyU3PGpifControlSWInput (CyBool_t *set*)

Function to set/clear the software controlled state machine input.

Description

The GPIF hardware supports one software driven internal input signal that can be used to control/direct the state machine functionality. This function is used to set the state of this input signal to a desired value.

Return value

None

Parameters

set	Whether to set (assert) the software input signal.

5.16.4.3 void CyU3PGpifDisable (CyBool_t *forceReload*)

Disables the GPIF state machine and hardware.

Description

This function is used to disable the GPIF state machine and hardware. If the forceReload parameter is set to true, the current configuration information is lost and needs to be restored using the CyU3PGpifLoad or equivalent functions. If the forceReload parameter is set to false, the GPIF configuration is retained and it is possible to restart

the it by calling the CyU3PGpifSMStart API.

Return value

None

See Also

CyU3PGpifConfigure

Parameters

forceReload	Whether a GPIF re-configuration is to be forced.

5.16.4.4 CyU3PReturnStatus_t CyU3PGpifGetSMState (uint8_t * *curState_p*)

Get the current state of the GPIF state machine.

Description

This function is used to fetch the current state of the GPIF state machine. This is primarily used to provide debug information.

Return value

CY_U3P_SUCCESS if the query is successful.

CY_U3P_ERROR_NOT_CONFIGURED if the state machine has not been configured as yet.

CY_U3P_ERROR_BAD_ARGUMENT if the return parameter has not been provided as part of the call.

See Also

CyU3PGpifSMStart
CyU3PGpifSMSwitch

Parameters

curState_p	Return parameter to be filled with current state information.

5.16.4.5 void CyU3PGpifInitAddrCounter (uint32_t *initValue*, uint32_t *limit*, CyBool_t *reload*, CyBool_t *upCount*, uint8_t *increment*)

Function to configure the GPIF address counter.

Description

This function is used to configure the GPIF address counter with the desired initial value, limit and counting mode.

Return value

None

Parameters

initValue	Initial value to start the counter from.
limit	Counter limit at which the counter stops.
reload	Whether to reload the counter when the limit is reached.
upCount	Whether to count upwards from the initial value.
increment	The value to be incremented/decremented from the counter at each step.

5.16.4.6 CyU3PReturnStatus_t CyU3PGpifInitComparator (CyU3PGpifComparatorType *type*, uint32_t *value*, uint32_t *mask*)

This function configures one of the comparators in the GPIF hardware.

Description

The GPIF hardware includes comparators that can be used to check if the control, address or data values match a desired pattern. There is a separate comparator for each class of signals. This function is used to configure and initialize one of these comparators as required.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_BAD_ARGUMENT if the type specified is invalid.

See Also

CyU3PGpifComparatorType

Parameters

type	The type of comparator to configure.
value	The value to compare the signals against.
mask	Mask that specifies which bits are to be used in the comparison.

5.16.4.7 void CyU3PGpifInitCtrlCounter (uint16_t *initValue*, uint16_t *limit*, CyBool_t *reload*, CyBool_t *upCount*, uint8_t *outputBit*)

Function to configure the GPIF control counter.

Description

The GPIF hardware includes a 16-bit control counter, one of whose bits can be connected to the CTRL[9] output from the device. This function is used to configure and initialize the control counter and to select the bit that should be connected to the CTRL[9] output. Please note that the specified bit location will be truncated to a value less than 16.

Return value

None

Parameters

initValue	Initial (reset) value for the counter.
limit	Value at which to stop the counter and flag an event.
reload	Whether to reload the counter and continue after limit is hit.
upCount	Whether to count upwards from the initial value.
outputBit	Selects counter bit to be connected to CTRL[9] output.

5.16.4.8 void CyU3PGpifInitDataCounter (uint32_t *initValue*, uint32_t *limit*, CyBool_t *reload*, CyBool_t *upCount*, uint8_t *increment*)

Function to configure the GPIF data counter.

Description

This function is used to configure the GPIF data counter with the desired initial value, limit and counting mode.

Return value

None

Parameters

initValue	Initial value to start the counter from.
limit	Counter limit at which the counter stops.
reload	Whether to reload the counter when the limit is reached.
upCount	Whether to count upwards from the initial value.
increment	The value to be incremented/decremented from the counter at each step.

5.16.4.9 CyU3PReturnStatus_t CyU3PGpifInitTransFunctions (uint16_t * *fnTable*)

Initialize the GPIF transition function registers.

Description

This function initializes the GPIF transition function registers with data provided by the GPIF designer tool.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_BAD_ARGUMENT if any of the parameters are invalid.

CY_U3P_ERROR_ALREADY_STARTED if the GPIF hardware is running.

See Also

CyU3PGpifConfigure

Parameters

fnTable	Pointer to the table (array) containing the values with which the registers should be initialized.

5.16.4.10 CyU3PReturnStatus_t CyU3PGpifLoad (const CyU3PGpifConfig_t * *conf*)

Function to program the user defined state machine into the GPIF registers.

Description

This function allows all of the configuration values corresponding to the user defined state machine to be loaded into the GPIF registers. The data to be passed as parameter to this function is received as output from the GPIF Designer tool.

Return value

CY_U3P_SUCCESS - if the configuration was successful.

CY_U3P_ERROR_NOT_SUPPORTED - if a 32 bit GPIF configuration is being used on a part that does not support this.

CY_U3P_ERROR_ALREADY_STARTED - if the GPIF hardware has already been programmed.

CY_U3P_ERROR_BAD_ARGUMENT - if the input to the API is inconsistent.

See Also

CyU3PGpifConfig_t

Parameters

conf	Pointer to GPIF configuration structure.

5.16.4.11 CyU3PReturnStatus_t CyU3PGpifOutputConfigure (uint8_t *ctrlPin,* CyU3PGpifOutput_t *opFlag,* CyBool_t *isActiveLow*)

Configure the functionality of a GPIF output pin.

Description

This function is used to configure the functionality of a GPIF output pin. This is primarily used to map a selected flag to a particular control output pin. The function configures the selected control pin as output, updates the polarity and connects the selected flag/signal to the pin.

Note

This configuration is likely to be overridden by any GPIF configuration API calls such as CyU3PGpifLoad or CyU3PGpifRegisterConfig.

Return value

CY_U3P_SUCCESS if the configuration is successful.

CY_U3P_ERROR_BAD_ARGUMENT if the control pin or output selected is invalid.

See Also

CyU3PGpifOutput_t

Parameters

ctrlPin	Control pin number to be configured.
opFlag	Selected output flag.
isActiveLow	Polarity: 0=Active high, 1=Active low.

5.16.4.12 CyU3PReturnStatus_t CyU3PGpifReadDataWords (uint32_t *threadIndex,* CyBool_t *selectThread,* uint32_t *numWords,* uint32_t * *buffer_p,* uint32_t *waitOption*)

Read a specified number of data words from the GPIF interface.

Description

The GPIF hardware supports a mode where data can be read from the P-port in terms of words of the specified interface width. The data can be read from any one of the four threads of data transfer across the P-port.

This function is used to read a specified number of data words into a buffer, one word at a time. Please note that each data word will be placed in the buffer after padding to 32 bits. A timeout period can be specified, and if any of the data words is not available within the specified period from the previous one, the operation will return with a timeout error.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_BAD_ARGUMENT if the thread number is invalid.

CY_U3P_ERROR_FAILURE if the operation is not permitted by the GPIF configuration.

Parameters

threadIndex	DMA thread index from which to read data.
selectThread	Whether the target thread should be enabled explicitly.
numWords	Number of words of data to read.
buffer_p	Buffer pointer into which the data should be read.
waitOption	Timeout duration to wait for data.

5.16.4.13 void CyU3PGpifRegisterCallback (CyU3PGpifEventCb_t *cbFunc*)

This function registers an event callback for the GPIF driver.

Description

The GPIF driver keeps track of GPIF related events and raises notifications to the application logic when required. This function is used to register the callback function that will be invoked to notify the application about GPIF events.

Return value

None

See Also

CyU3PGpifEventCb_t
CyU3PGpifEventType

Parameters

cbFunc	Pointer to callback function.

5.16.4.14 CyU3PReturnStatus_t CyU3PGpifRegisterConfig (uint16_t *numRegs,* uint32_t *regData[][2]*)

Initialize the GPIF configuration registers.

Description

This is an alternate flavor of the GPIF register configuration function that can be used when only a small subset of the registers needs to be initialized. The input is accepted in the form of a two-columned matrix, column 0 representing the register address to be initialized and column 1 representing the value to be written into this register.

Please note that the registers will be initialized in the same order they are provided in the array, and that it is possible to write multiple times to the same register.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_NOT_SUPPORTED - if a 32 bit GPIF configuration is being used on a part that does not support this.

CY_U3P_ERROR_BAD_ARGUMENT if any of the parameters are invalid.

CY_U3P_ERROR_ALREADY_STARTED if the GPIF hardware is running.

See Also

CyU3PGpifConfigure

Parameters

numRegs	Number of registers to configure (size of the matrix).
regData	Reference to array of {address, data} tuples corresponding to GPIF registers to initialize.

5.16.4.15 void CyU3PGpifRegisterSMIntrCallback (CyU3PGpifSMIntrCb_t *cb*)

Register a callback function for fast notification of GPIF state machine interrupts.

Description

This function registers a callback function that will be invoked from interrupt context when a GPIF state machine interrupt is detected (caused by the INTR_CPU action in the GPIF state machine).

This callback provides a fast notification of the interrupt and is not subject to thread switching delays. Please note that API calls that require a mutex get or equivalent cannot be directly called from this callback function.

Return value

None

See Also

CyU3PGpifSMIntrCb_t

Parameters

cb	Callback function pointer.

5.16.4.16 CyU3PReturnStatus_t CyU3PGpifSMControl (CyBool_t *pause*)

Function to pause/resume the GPIF state machine.

Description

The GPIF state machine continuously functions on the basis of configuration values that are set using the CyU3P-GpifWaveformLoad API. While the state machine normally functions without any software control, it is possible for the software to stop/start the functioning of the state machine at will. This function allows the application to either pause or resume the state machine.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_NOT_CONFIGURED if the state machine has not been configured.

Parameters

pause	Whether to pause the state machine or resume it.

5.16.4.17 CyU3PReturnStatus_t CyU3PGpifSMStart (uint8_t *stateIndex*, uint8_t *initialAlpha*)

Start the waveform state machine from the specified state.

Description

This function starts off the GPIF state machine from the specified state index. Please note that this function should only be used to start the state machine from an idle state. The CyU3PGpifSMSwitch function should be used to switch from one state to another in mid-execution.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_NOT_CONFIGURED if GPIF has not been configured as yet.

CY_U3P_ERROR_ALREADY_STARTED if the state machine is already active.

See Also

CyU3PGpifSMSwitch

Parameters

stateIndex	State from which to start execution. This should be 0 in most cases.
initialAlpha	Initial alpha values to start GPIF state machine operation with.

5.16.4.18 CyU3PReturnStatus_t CyU3PGpifSMSwitch (uint16_t *fromState*, uint16_t *toState*, uint16_t *endState*, uint8_t *initialAlpha*, uint32_t *switchTimeout*)

This function is used to start GPIF state machine execution from a desired state.

Description

This function allows the caller to switch to a desired state and continue GPIF state machine execution from there. The toState parameter specifies the state to initiate operation from.

The fromState parameter can be used to ensure that the transition to toState happens only when the state machine is in a well defined idle state. If a valid state id (0 - 255) is passed as the fromState, the transition is only allowed from that state index. If not, the state machine is immediately switched to the toState.

The endState can be used to obtain a notification when the state machine execution has reached the designated end state. Again, this functionality is only valid if a valid endState value is passed in.

The switchTimeout specifies the amount of time to wait for the transition to the desired toState. This is only meaningful if a fromState is specified, and the timeout value is specified in terms of GPIF hardware clock cycles.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_BAD_ARGUMENT if any of the parameters are invalid.

See Also

CyU3PGpifSMStart

Parameters

fromState	The state from which to do the switch to the desired state.
toState	The state to which to transition from fromState.
endState	The end state for this execution path.
initialAlpha	Initial Alpha values to use when switching states.
switchTimeout	Timeout setting for the switch operation in GPIF clock cycles.

5.16.4.19 CyU3PReturnStatus_t CyU3PGpifSocketConfigure (uint8_t *threadIndex*, CyU3PDmaSocketId_t *socketNum*, uint16_t *watermark*, CyBool_t *flagOnData*, uint8_t *burst*)

Function to select an active socket on the P-port and to configure it.

Description

The GPIF hardware allows 4 different sockets on the P-port to be accessed at a time by supporting 4 independent DMA threads. The active socket for each thread and its properties can be selected by the user at run-time. This should be done in software only in the case where it is not being done through the PP registers or the state machine itself.

This function allows the user to select and configure the active socket in the case where software is responsible for these actions. The API will respond with an error if the hardware is taking care of socket configurations.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_FAILURE if the socket selection and configuration is being done automatically.

CY_U3P_ERROR_BAD_ARGUMENT if one or more of the parameters are out of range.

Parameters

threadIndex	Thread index whose active socket is to be configured.
socketNum	The socket to be associated with this thread.

watermark	Watermark level for this socket in number of 4-byte words.
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.
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.

5.16.4.20 CyU3PReturnStatus_t CyU3PGpifWaveformLoad (uint8_t *firstState*, uint16_t *stateCnt*, uint8_t * *stateDataMap*, CyU3PGpifWaveData * *transitionData*)

Initialize the GPIF state machine table.

Description

The GPIF state machine is programmed in the form of a set of waveform table entries. This function is used to take the output state machine data from the designer tool and to initialize the state machine based on these values.

Note that the state data is generated in the form of a transition data array and a state index mapping table by the 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.

Return value

CY_U3P_SUCCESS if successful.

CY_U3P_ERROR_BAD_ARGUMENT if any of the parameters are invalid.

CY_U3P_ERROR_ALREADY_STARTED if the GPIF hardware is running.

See Also

CyU3PGpifWaveData

Parameters

firstState	The first state to be initialized in this function call.
stateCnt	Number of states to be initialized with data.
stateDataMap	Table that maps state indices to the descriptor table indices.
transitionData	Table containing the transition information for various states.

5.16.4.21 CyU3PReturnStatus_t CyU3PGpifWriteDataWords (uint32_t *threadIndex*, CyBool_t *selectThread*, uint32_t *numWords*, uint32_t * *buffer_p*, uint32_t *waitOption*)

Write a specified number of data words to the GPIF interface.

Description

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 value

CY_U3P_SUCCESS if successful.

Parameters

threadIndex	DMA thread index through which to write data.
selectThread	Whether the target thread should be enabled explicitly.
numWords	Number of words of data to write.
buffer_p	Pointer to buffer containing the data.
waitOption	Timeout duration to wait for data sending.

5.17 firmware/u3p_firmware/inc/cyu3gpio.h File Reference

The GPIO manager module is responsible for handling general purpose IO pins on the FX3 device.

```
#include "cyu3types.h"
#include "cyu3system.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PGpioSimpleConfig_t

  Configuration information for simple GPIOs.
- struct CyU3PGpioComplexConfig_t

  Configuration information for complex GPIO pins.

Typedefs

- typedef enum CyU3PGpioComplexMode_t CyU3PGpioComplexMode_t

  Enumerated list of all GPIO complex modes.
- typedef enum CyU3PGpioTimerMode_t CyU3PGpioTimerMode_t

  List of possible clock sources for GPIO timers.
- typedef enum CyU3PGpioIntrMode_t CyU3PGpioIntrMode_t

  List of interrupt modes supported by FX3 GPIOs.
- typedef struct CyU3PGpioSimpleConfig_t CyU3PGpioSimpleConfig_t

  Configuration information for simple GPIOs.
- typedef struct CyU3PGpioComplexConfig_t CyU3PGpioComplexConfig_t

  Configuration information for complex GPIO pins.
- typedef void(∗ CyU3PGpioIntrCb_t)(uint8_t gpioId)

  GPIO callback function type.

Enumerations

- enum CyU3PGpioComplexMode_t { CY_U3P_GPIO_MODE_STATIC = 0, CY_U3P_GPIO_MODE_TOGGLE, CY_U3P_GPIO_MODE_SAMPLE_NOW, CY_U3P_GPIO_MODE_PULSE_NOW, CY_U3P_GPIO_MODE_PULSE, CY_U3P_GPIO_MODE_PWM, CY_U3P_GPIO_MODE_MEASURE_LO-

W, CY_U3P_GPIO_MODE_MEASURE_HIGH, CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE, CY_U3P_GPIO_MODE_MEASURE_HIGH_ONCE, CY_U3P_GPIO_MODE_MEASURE_NEG, CY_U3P_GPIO_MODE_MEASURE_POS, CY_U3P_GPIO_MODE_MEASURE_ANY, CY_U3P_GPIO_MODE_MEASURE_NEG_ONCE, CY_U3P_GPIO_MODE_MEASURE_POS_ONCE, CY_U3P_GPIO_MODE_MEASURE_ANY_ONCE }

Enumerated list of all GPIO complex modes.

- enum CyU3PGpioTimerMode_t { CY_U3P_GPIO_TIMER_SHUTDOWN = 0, CY_U3P_GPIO_TIMER_HIGH_FREQ, CY_U3P_GPIO_TIMER_LOW_FREQ, CY_U3P_GPIO_TIMER_STANDBY_FREQ, CY_U3P_GPIO_TIMER_POS_EDGE, CY_U3P_GPIO_TIMER_NEG_EDGE, CY_U3P_GPIO_TIMER_ANY_EDGE, CY_U3P_GPIO_TIMER_RESERVED }

  List of possible clock sources for GPIO timers.
- enum CyU3PGpioIntrMode_t { CY_U3P_GPIO_NO_INTR = 0, CY_U3P_GPIO_INTR_POS_EDGE, CY_U3P_GPIO_INTR_NEG_EDGE, CY_U3P_GPIO_INTR_BOTH_EDGE, CY_U3P_GPIO_INTR_LOW_LEVEL, CY_U3P_GPIO_INTR_HIGH_LEVEL, CY_U3P_GPIO_INTR_TIMER_THRES, CY_U3P_GPIO_INTR_TIMER_ZERO }

  List of interrupt modes supported by FX3 GPIOs.

Functions

- CyU3PReturnStatus_t CyU3PGpioInit (CyU3PGpioClock_t *clk_p, CyU3PGpioIntrCb_t irq)

  Initializes the GPIO Manager.
- CyU3PReturnStatus_t CyU3PGpioDeInit (void)

  De-initializes the GPIO Manager.
- CyU3PReturnStatus_t CyU3PGpioSetSimpleConfig (uint8_t gpioId, CyU3PGpioSimpleConfig_t *cfg_p)

  Configures a simple GPIO.
- CyU3PReturnStatus_t CyU3PGpioSetComplexConfig (uint8_t gpioId, CyU3PGpioComplexConfig_t *cfg_p)

  Configures a complex GPIO pin.
- CyU3PReturnStatus_t CyU3PGpioDisable (uint8_t gpioId)

  Disables a GPIO pin.
- CyU3PReturnStatus_t CyU3PGpioGetValue (uint8_t gpioId, CyBool_t *value_p)

  Query the state of GPIO input.
- CyU3PReturnStatus_t CyU3PGpioSimpleGetValue (uint8_t gpioId, CyBool_t *value_p)

  Query the state of simple GPIO input.
- CyU3PReturnStatus_t CyU3PGpioSetValue (uint8_t gpioId, CyBool_t value)

  Update the state of a GPIO output.
- CyU3PReturnStatus_t CyU3PGpioSimpleSetValue (uint8_t gpioId, CyBool_t value)

  Update the state of a simple GPIO output pin.
- CyU3PReturnStatus_t CyU3PGpioGetIOValues (uint32_t *gpioVal0_p, uint32_t *gpioVal1_p)

  Get the current state of all GPIOs.
- CyU3PReturnStatus_t CyU3PGpioComplexUpdate (uint8_t gpioId, uint32_t threshold, uint32_t period)

  Update the timer period and threshold associated with a complex GPIO.
- CyU3PReturnStatus_t CyU3PGpioComplexGetThreshold (uint8_t gpioId, uint32_t *threshold_p)

  Read the threshold value associated with a complex GPIO.
- CyU3PReturnStatus_t CyU3PGpioComplexSampleNow (uint8_t gpioId, uint32_t *value_p)

  Sample the GPIO timer value at an instant.
- CyU3PReturnStatus_t CyU3PGpioComplexPulseNow (uint8_t gpioId, uint32_t threshold)

  Drive a pulse on the specified complex GPIO output immediately.
- CyU3PReturnStatus_t CyU3PGpioComplexPulse (uint8_t gpioId, uint32_t threshold)

  Configures a complex GPIO to drive an output pulse.

- CyU3PReturnStatus_t CyU3PGpioComplexMeasureOnce (uint8_t gpioId, CyU3PGpioComplexMode_t pinMode)

  Initiate an input signal duration measurement.
- CyU3PReturnStatus_t CyU3PGpioComplexWaitForCompletion (uint8_t gpioId, uint32_t *threshold_p, CyBool_t isWait)

  Wait for the completion of MeasureOnce, Pulse and PulseNow operations.
- void CyU3PRegisterGpioCallBack (CyU3PGpioIntrCb_t gpioIntrCb)

  Register a callback for notification of GPIO events.

5.17.1 Detailed Description

The GPIO manager module is responsible for handling general purpose IO pins on the FX3 device. **Description**

Almost any pin of the FX3 device can be used as a General Purpose I/O (GPIO), subject to the condition that it is not being used for any other purpose. The FX3 device supports two classes of GPIOs - Simple and Complex.

Simple GPIOs provide software controlled and observable input and output capability only. The only additional functionality supported on these are interrupt detection on edge or signal level. Any unused pin of the FX3 can be mapped and used as a simple GPIO.

Complex GPIOs are value-added blocks that support additional functions like PWM output, pulsing, time measurements and timers. A maximum of 8 pins on the FX3 can be configured as complex GPIOs, and each of these pins should have a different remainder module 8.

As each GPIO is controlled through a separate set of registers on the FX3 device, they can only be configured individually and multiple GPIOs cannot be updated simultaneously.

5.17.2 Typedef Documentation

5.17.2.1 typedef struct CyU3PGpioComplexConfig_t CyU3PGpioComplexConfig_t

Configuration information for complex GPIO pins.

Description

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

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

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioSimpleConfig_t
CyU3PGpioComplexMode_t
CyU3PGpioTimerMode_t
CyU3PGpioIntrMode_t

5.17.2.2 typedef enum CyU3PGpioComplexMode_t CyU3PGpioComplexMode_t

Enumerated list of all GPIO complex modes.

Description

This enumeration lists the complex GPIO modes supported by the FX3 device.

Of these, the CY_U3P_GPIO_MODE_SAMPLE_NOW, CY_U3P_GPIO_MODE_PULSE_NOW, CY_U3P_GPIO_-MODE_PULSE, CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE, CY_U3P_GPIO_MODE_MEASURE_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 directly selected. To use any of these modes, the complex GPIO should be configured with CY_U3P_GPIO_MODE_STATIC configuration; and then special GPIO APIs need to be used for completing the desired operations.

See Also

CyU3PGpioTimerMode_t
CyU3PGpioIntrMode_t

5.17.2.3 typedef void(* CyU3PGpioIntrCb_t)(uint8_t gpioId)

GPIO callback function type.

Description

The GPIO manager in FX3 firmware supports event notifications when a GPIO related interrupt is detected. This data type defines the signature of the callback function that can be registered to receive the event notifications.

See Also

CyU3PRegisterGpioCallBack

5.17.2.4 typedef enum CyU3PGpioIntrMode_t CyU3PGpioIntrMode_t

List of interrupt modes supported by FX3 GPIOs.

Description

GPIOs on the FX3 device can be configured to interrupt the firmware under various conditions. For simple GPIOs, the interrupt conditions are based on specific input levels or edge detection. Complex GPIOs support additional interrupts based on the timer period and the threshold value.

This enumerated type lists the various interrupt modes supported by FX3 GPIOs.

See Also

CyU3PGpioComplexMode_t
CyU3PGpioTimerMode_t

5.17.2.5 typedef struct CyU3PGpioSimpleConfig_t CyU3PGpioSimpleConfig_t

Configuration information for simple GPIOs.

Description

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

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

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

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

See Also

CyU3PGpioIntrMode_t
CyU3PGpioSetSimpleConfig

5.17.2.6 typedef enum CyU3PGpioTimerMode_t CyU3PGpioTimerMode_t

List of possible clock sources for GPIO timers.

Description

The timer blocks associated with Complex GPIOs can be clocked internally, or based on the input signal connected to the corresponding pin. This enumerated type lists the various clocking options for these GPIO timers.

See Also

CyU3PGpioClock_t
CyU3PGpioComplexMode_t
CyU3PGpioIntrMode_t

5.17.3 Enumeration Type Documentation

5.17.3.1 enum CyU3PGpioComplexMode_t

Enumerated list of all GPIO complex modes.

Description

This enumeration lists the complex GPIO modes supported by the FX3 device.

Of these, the CY_U3P_GPIO_MODE_SAMPLE_NOW, CY_U3P_GPIO_MODE_PULSE_NOW, CY_U3P_GPIO_-MODE_PULSE, CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE, CY_U3P_GPIO_MODE_MEASURE_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 directly selected. To use any of these modes, the complex GPIO should be configured with CY_U3P_GPIO_MODE_STATIC configuration; and then special GPIO APIs need to be used for completing the desired operations.

See Also

CyU3PGpioTimerMode_t
CyU3PGpioIntrMode_t

Enumerator

CY_U3P_GPIO_MODE_STATIC Drives simple static values on GPIO.

CY_U3P_GPIO_MODE_TOGGLE Toggles the output when timer = threshold.

CY_U3P_GPIO_MODE_SAMPLE_NOW Read current timer value into threshold. This is a one time operation and resets mode to static. This mode should not be set by the configure function. This will be done by the CyU3PGpioComplexSampleNow API.

CY_U3P_GPIO_MODE_PULSE_NOW Mode used for driving a pulse out on a specified GPIO pin at a specified time. This mode should not be set directly by the configure function. This will be done by the CyU3PGpioComplexPulseNow API.

CY_U3P_GPIO_MODE_PULSE This is similar to the PULSE_NOW mode, and is used to drive a pulse out when the timer associated with the GPIO reaches 0. This mode should not be set by the configure function and will be done by the CyU3PGpioComplexPulse API.

CY_U3P_GPIO_MODE_PWM Drive a Pulse Width Modulated (PWM) waveform out on the specified GPIO. The ON and OFF times for the signal are defined by the timer period and the threshold value. This is a continuous operation.

CY_U3P_GPIO_MODE_MEASURE_LOW This mode is used to measure the length of time for which a specified input pin is low. This is a continuous operation which provides the interval duration every time there is a positive edge on the pin.

CY_U3P_GPIO_MODE_MEASURE_HIGH This mode is similar to the MEASURE_LOW mode and is used to measure the duration for which an input pin is driver high.

CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE This mode is similar to the MEASURE_LOW mode, except that the pulse duration is measured only once. The mode of the pin will revert to STATIC at the end of the first low pulse. This mode cannot be directly selected and is used internally by the CyU3PGpioComplexMeasureOnce API.

CY_U3P_GPIO_MODE_MEASURE_HIGH_ONCE This mode is similar to the MEASURE_LOW_ONCE mode, and is used to do a single duration measurement for a high pulse. This mode is also used internally by the CyU3PGpioComplexMeasureOnce API.

CY_U3P_GPIO_MODE_MEASURE_NEG This mode is used to measure the delay until the specified GPIO input goes low. This is a continuous operation, where the Threshold value is updated with the current timer value at the point where the signal goes low.

CY_U3P_GPIO_MODE_MEASURE_POS This mode is similar to the MEASURE_NEG mode, and is a continuous mode which can be used to measure the delay until the GPIO input is driven high.

CY_U3P_GPIO_MODE_MEASURE_ANY This mode is similar to the MEASURE_NEG and MEASURE_POS modes; and is used to measure the delay until there is any change in the GPIO input signal.

CY_U3P_GPIO_MODE_MEASURE_NEG_ONCE This mode is similar to the MEASURE_NEG mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

CY_U3P_GPIO_MODE_MEASURE_POS_ONCE This mode is similar to the MEASURE_POS mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

CY_U3P_GPIO_MODE_MEASURE_ANY_ONCE This mode is similar to the MEASURE_POS mode, except that only a single delay measurement is performed. This mode cannot be directly selected, and is used internally by the CyU3PGpioComplexMeasureOnce API.

5.17.3.2 enum CyU3PGpioIntrMode_t

List of interrupt modes supported by FX3 GPIOs.

Description

GPIOs on the FX3 device can be configured to interrupt the firmware under various conditions. For simple GPIOs, the interrupt conditions are based on specific input levels or edge detection. Complex GPIOs support additional interrupts based on the timer period and the threshold value.

This enumerated type lists the various interrupt modes supported by FX3 GPIOs.

See Also

CyU3PGpioComplexMode_t
CyU3PGpioTimerMode_t

Enumerator

CY_U3P_GPIO_NO_INTR GPIO interrupt is unused.

CY_U3P_GPIO_INTR_POS_EDGE Interrupt is triggered for positive edge of input.

CY_U3P_GPIO_INTR_NEG_EDGE Interrupt is triggered for negative edge of input.

CY_U3P_GPIO_INTR_BOTH_EDGE Interrupt is triggered on either edge of input.

CY_U3P_GPIO_INTR_LOW_LEVEL Interrupt is triggered when the input value is low.

CY_U3P_GPIO_INTR_HIGH_LEVEL Interrupt is triggered when the input value is high.

CY_U3P_GPIO_INTR_TIMER_THRES Interrupt is triggered when the timer crosses the threshold. Valid only for complex GPIO pins.

CY_U3P_GPIO_INTR_TIMER_ZERO Interrupt is triggered when the timer reaches zero. Valid only for complex GPIO pins.

5.17.3.3 enum CyU3PGpioTimerMode_t

List of possible clock sources for GPIO timers.

Description

The timer blocks associated with Complex GPIOs can be clocked internally, or based on the input signal connected to the corresponding pin. This enumerated type lists the various clocking options for these GPIO timers.

See Also

CyU3PGpioClock_t
CyU3PGpioComplexMode_t
CyU3PGpioIntrMode_t

Enumerator

CY_U3P_GPIO_TIMER_SHUTDOWN Timer not in use (no clock applied).

CY_U3P_GPIO_TIMER_HIGH_FREQ Use GPIO fast clock.

CY_U3P_GPIO_TIMER_LOW_FREQ Use GPIO slow clock.

CY_U3P_GPIO_TIMER_STANDBY_FREQ Use FX3 back-up clock (32 KHz).

CY_U3P_GPIO_TIMER_POS_EDGE Timer updates on positive edge of input.

CY_U3P_GPIO_TIMER_NEG_EDGE Timer updates on negative edge of input.

CY_U3P_GPIO_TIMER_ANY_EDGE Timer updates on either edge of input.

CY_U3P_GPIO_TIMER_RESERVED Reserved for future use.

5.17.4 Function Documentation

5.17.4.1 CyU3PReturnStatus_t CyU3PGpioComplexGetThreshold (uint8_t *gpiold*, uint32_t * *threshold_p*)

Read the threshold value associated with a complex GPIO.

Description

This function reads the current threshold value associated with a complex GPIO. This function is used in the various MEASURE modes of the GPIO to get the measured pulse/signal duration.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the IO specified is not valid.

CY_U3P_ERROR_NULL_POINTER - If the threshold_p is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - If the GPIO is not enabled.

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioComplexUpdate
CyU3PGpioComplexMeasureOnce

Parameters

gpioId	The complex GPIO to sample.
threshold_p	Returns the current threshold value for the GPIO.

5.17.4.2 CyU3PReturnStatus_t CyU3PGpioComplexMeasureOnce (uint8_t *gpioId*, CyU3PGpioComplexMode_t *pinMode*)

Initiate an input signal duration measurement.

Description

The complex GPIOs on FX3 are capable of measuring various duration parameters on the input signal.

CY_U3P_GPIO_MODE_MEASURE_LOW_ONCE : Low period for input.

CY_U3P_GPIO_MODE_MEASURE_HIGH_ONCE : High period for input.

CY_U3P_GPIO_MODE_MEASURE_NEG_ONCE : Time to negative edge.

CY_U3P_GPIO_MODE_MEASURE_POS_ONCE : Time to positive edge.

CY_U3P_GPIO_MODE_MEASURE_ANY_ONCE : Time to the next edge (transition).

This API is used to initiate a single measurement of the desired timing parameter on a specified GPIO. This API can only be used when the GPIO is in the CY_U3P_GPIO_MODE_STATIC mode and the timer is operational.

The API does not wait for the transition that ends the measurement, but returns immediately. The CyU3PGpioComplexWaitForCompletion API can be used to wait until the condition that ends the measurement is reached.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If IO or the mode specified is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - If the IO is not enabled.

CY_U3P_ERROR_NOT_SUPPORTED - If the GPIO is not configured correctly for this operation.

See Also

CyU3PGpioComplexMode_t
CyU3PGpioSetComplexConfig

Parameters

gpioId	The complex GPIO to measure.
pinMode	Type of measurement to be performed.

5.17.4.3 CyU3PReturnStatus_t CyU3PGpioComplexPulse (uint8_t *gpioId*, uint32_t *threshold*)

Configures a complex GPIO to drive an output pulse.

Description

This API is a delayed version of the CyU3PGpioComplexPulseNow where the start of the pulse output happens when the GPIO timer reaches 0.

The pulse duration is again determined by the threshold value, and this function returns as soon as it has configured the GPIO to drive the pulse out.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the IO or threshold is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - If the IO is not enabled.

CY_U3P_ERROR_NOT_SUPPORTED - If the GPIO is not configured correctly for this operation.

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioComplexPulseNow

Parameters

gpioId	The complex GPIO to pulse.
threshold	Updates the threshold value used for setting the pulse duration.

5.17.4.4 CyU3PReturnStatus_t CyU3PGpioComplexPulseNow (uint8_t *gpioId*, uint32_t *threshold*)

Drive a pulse on the specified complex GPIO output immediately.

Description

This function is used to drive a pulse out on the specified complex GPIO, starting immediately. The timer value associated with the GPIO is set to 0 at the beginning of the pulse. The pulse duration is determined by the threshold value and the signal is automatically driven low once the threshold is reached.

This API does not wait until the end of the pulse, but returns as soon as the pulse has been started. The mode for the GPIO is reverted to STATIC as soon as the pulse is completed.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the IO or threshold is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - If the IO is not enabled.

CY_U3P_ERROR_NOT_SUPPORTED - If the GPIO is not configured correctly for this operation.

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioComplexPulse

Parameters

gpioId	The complex GPIO to pulse.
threshold	Updates the threshold value used for setting the pulse duration.

5.17.4.5 CyU3PReturnStatus_t CyU3PGpioComplexSampleNow (uint8_t *gpioId*, uint32_t * *value_p*)

Sample the GPIO timer value at an instant.

Description

This function is used to sample the current value of the GPIO timer at an instant. This can be used to estimate the elapsed time between multiple events processed by the firmware.

To use this feature, the GPIO should be configured as CY_U3P_GPIO_MODE_STATIC and the timer should be

operational.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the IO is invalid.

CY_U3P_ERROR_NULL_POINTER - If the value_p is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - If the GPIO is not enabled.

CY_U3P_ERROR_NOT_SUPPORTED - Invalid configuration.

See Also

CyU3PGpioSetComplexConfig

Parameters

gpioId	The complex GPIO to sample.
value_p	Returns the current value of the timer.

5.17.4.6 CyU3PReturnStatus_t CyU3PGpioComplexUpdate (uint8_t *gpioId*, uint32_t *threshold*, uint32_t *period*)

Update the timer period and threshold associated with a complex GPIO.

Description

The function updates the timer period and threshold value associated with a complex GPIO. This operation is only permitted if the corresponding is configured in the PWM or STATIC modes.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the IO pin is invalid.

See Also

CyU3PGpioSetComplexConfig
CyU3PGpioComplexGetThreshold

Parameters

gpioId	The complex GPIO to sample.
threshold	New timer threshold value to be updated.
period	New timer period value to be updated.

5.17.4.7 CyU3PReturnStatus_t CyU3PGpioComplexWaitForCompletion (uint8_t *gpioId*, uint32_t * *threshold_p*, CyBool_t *isWait*)

Wait for the completion of MeasureOnce, Pulse and PulseNow operations.

Description

This function checks or wait for the completion of a previously initiated CyU3PGpioComplexPulse, CyU3PGpioComplexPulseNow or CyU3PGpioComplexMeasureOnce operation.

If the isWait option is set to false, the API only checks for completion and returns a TIMEOUT error if it is not

complete. If the isWait option is set to true, the API enters a busy-wait loop that is only terminated when the GPIO operation is complete.

Note

A busy wait is used to wait for the completion of the GPIO operation. Please use this with care, as this can affect the operation of other blocks and drivers in the FX3 firmware.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the Drive strength requested is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - If the IO is not enabled.

CY_U3P_ERROR_TIMEOUT - If the operation is not complete at the time of checking.

CY_U3P_ERROR_NOT_SUPPORTED - If the GPIO is not configured correctly for this operation.

See Also

CyU3PGpioComplexPulse
CyU3PGpioComplexPulseNow
CyU3PGpioComplexMeasureOnce

Parameters

gpioId	The complex GPIO to wait for completion.
threshold_p	The current threshold value after completion.
isWait	Whether to wait or just check for completion.

5.17.4.8 CyU3PReturnStatus_t CyU3PGpioDeInit (void)

De-initializes the GPIO Manager.

Description

De-initialize the GPIO manager in FX3 firmware and power off the GPIO block on the FX3 device.

Return value

CY_U3P_SUCCESS - If the GPIO de-init is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block was not previously initialized.

See Also

CyU3PGpioInit

5.17.4.9 CyU3PReturnStatus_t CyU3PGpioDisable (uint8_t *gpioId*)

Disables a GPIO pin.

Description

This function is used to disable a GPIO pin which was previously configured as a simple or a complex GPIO.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the GPIO id is invalid.

See Also

CyU3PGpioSetSimpleConfig
CyU3PGpioSetComplexConfig

Parameters

gpioId	GPIO ID to be disabled

5.17.4.10 CyU3PReturnStatus_t CyU3PGpioGetIOValues (uint32_t * *gpioVal0_p*, uint32_t * *gpioVal1_p*)

Get the current state of all GPIOs.

Description

This API returns the state of all FX3 IO pins. The state information is returned in the form of a bit vector, where each bit represents the state of the corresponding GPIO pin. The retrieved information is valid only for pins configured as GPIOs.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

See Also

CyU3PGpioGetValue
CyU3PGpioSimpleGetValue

Parameters

gpioVal0_p	Bit vector that represents the states of GPIO 31 : 00.
gpioVal1_p	Bit vector that represents the states of GPIO 60 : 32. The upper three bits are unused and reserved.

5.17.4.11 CyU3PReturnStatus_t CyU3PGpioGetValue (uint8_t *gpioId*, CyBool_t * *value_p*)

Query the state of GPIO input.

Description

Get the current state of an input pin configured as a simple or a complex GPIO. This function can also retrieve the last updated outValue for an output pin.

This API can be used to get the current state of an input pin, even if it is configured as an interrupt source.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the GPIO id is invalid.

CY_U3P_ERROR_NULL_POINTER - If value_p is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - If the pin is not selected as a GPIO in the IOMATRIX.

See Also

CyU3PGpioSetSimpleConfig
CyU3PGpioSetComplexConfig
CyU3PGpioSimpleGetValue

CyU3PGpioSetValue
CyU3PGpioGetIOValues

Parameters

gpioId	GPIO id to be queried.
value_p	Output parameter that will be filled with the GPIO value.

5.17.4.12 CyU3PReturnStatus_t CyU3PGpioInit (CyU3PGpioClock_t * *clk_p*, CyU3PGpioIntrCb_t *irq*)

Initializes the GPIO Manager.

Description

This function initializes the GPIO manager in the FX3 firmware and powers up the GPIO block on the FX3 device. All other GPIO functions in the SDK can be used only after this function is called.

The clock parameters for the GPIO block are defined through this function. Three different clocks are associated with the FX3 GPIO clock.

The FAST clock is derived from the FX3 system clock, and its frequency is determined by the clkSrc and fastClkDiv parameters. The SLOW clock is derived from the FAST clock and its frequency is determined by the slowClkDiv value. The operating clock for various complex GPIO timers can be selected from among the FAST clock, the SLOW clock and a fixed 32 KHz frequency.

All simple GPIOs function (are updated or sampled) based on a separate clock which is also derived from the FAST clock. The frequency of this clock is determined by the simpleDiv value.

This function also allows for a GPIO interrupt callback to be registered. This function will be called when any GPIO related interrupt is raised by the device, and will receive the GPIO ID as a parameter. Please note that this interrupt call is made in interrupt context, which means that any blocking API calls cannot be made from this callback.

Return value

CY_U3P_SUCCESS - If the GPIO initialization is successful.

CY_U3P_ERROR_ALREADY_STARTED - If the GPIO has already been initialized.

CY_U3P_ERROR_NULL_POINTER - If clk_p is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - If an incorrect / invalid clock value is passed.

See Also

CyU3PGpioClock_t
CyU3PGpioIntrCb_t
CyU3PGpioDeInit

Parameters

clk_p	Clock settings for the GPIO block.
irq	GPIO interrupt callback function.

5.17.4.13 CyU3PReturnStatus_t CyU3PGpioSetComplexConfig (uint8_t *gpioId*, CyU3PGpioComplexConfig_t * *cfg_p*)

Configures a complex GPIO pin.

Description

This function is used to configure and enable a complex GPIO pin. This function needs to be called before using the complex GPIO pin.

A maximum of 8 complex GPIOs can be used on FX3, and the assignment of pin to complex GPIO is done on a MODULO 8 basis. i.e., only one of the pins 0, 8, 16, .. 56 can be used as complex GPIO and so on.

It is possible to use the timer associated with a complex GPIO without actually configuring or using the corresponding pin as a GPIO. In this case, all of the inputEn, driveLowEn and driveHighEn values can be set to CyFalse.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If any of the parameters are incorrect/invalid.

CY_U3P_ERROR_NULL_POINTER - If cfg_p is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - If the pin has not been selected as a complex GPIO in the IO matrix.

See Also

CyU3PGpioComplexMode_t
CyU3PGpioComplexConfig_t
CyU3PGpioSetSimpleConfig
CyU3PGpioDisable
CyU3PGpioComplexUpdate
CyU3PGpioComplexGetThreshold
CyU3PGpioComplexWaitForCompletion
CyU3PGpioComplexSampleNow
CyU3PGpioComplexPulseNow
CyU3PGpioComplexPulse
CyU3PGpioComplexMeasureOnce

Parameters

gpioId	GPIO id to be modified
cfg_p	Desired GPIO configuration.

5.17.4.14 CyU3PReturnStatus_t CyU3PGpioSetSimpleConfig (uint8_t *gpioId*, CyU3PGpioSimpleConfig_t * *cfg_p*)

Configures a simple GPIO.

Description

This function configures and enables a simple GPIO pin. The pin being configured needs to have been selected as a simple GPIO through the CyU3PDeviceConfigureIOMatrix or CyU3PDeviceGpioOverride API calls.

The operating parameters for the GPIO pin as passed in as parameters.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If any of the parameters are incorrect/invalid.

CY_U3P_ERROR_NULL_POINTER - If cfg_p is NULL.

CY_U3P_ERROR_NOT_CONFIGURED - If the pin has not been selected as a GPIO in the IO matrix configuration.

See Also

CyU3PGpioSimpleConfig_t
CyU3PDeviceConfigureIOMatrix
CyU3PDeviceGpioOverride
CyU3PGpioDisable

CyU3PGpioSimpleSetValue
CyU3PGpioSimpleGetValue
CyU3PGpioSetValue
CyU3PGpioGetValue

Parameters

gpioId	Id of the pin being selected as a GPIO.
cfg_p	Desired GPIO configuration.

5.17.4.15 CyU3PReturnStatus_t CyU3PGpioSetValue (uint8_t *gpioId*, CyBool_t *value*)

Update the state of a GPIO output.

Description

This function updates the state of a GPIO output pin. The driveLowEn and driveHighEn values set at the time of GPIO configuration will determine whether the pin is actually driven with the desired value or tri-stated.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the GPIO id is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - If the pin is not selected as a GPIO in the IOMATRIX.

See Also

CyU3PGpioSetSimpleConfig
CyU3PGpioSetComplexConfig
CyU3PGpioSimpleSetValue
CyU3PGpioGetValue

Parameters

gpioId	GPIO id to be modified.
value	Value to set on the GPIO pin.

5.17.4.16 CyU3PReturnStatus_t CyU3PGpioSimpleGetValue (uint8_t *gpioId*, CyBool_t * *value_p*)

Query the state of simple GPIO input.

Description

Get the current signal state of a simple GPIO configured as an input signal. The CyU3PGpioGetValue API supports all kinds of GPIOs and has a fair amount of conditions checks that slow the operation down. This dedicated API works only on simple GPIOs and will give better GPIO access performance.

This API can be used to get the current state of an input pin, even if it is configured as an interrupt source.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the GPIO id is invalid.

CY_U3P_ERROR_NULL_POINTER - If value_p is NULL.

See Also

CyU3PGpioSetSimpleConfig
CyU3PGpioSimpleSetValue
CyU3PGpioGetValue
CyU3PGpioGetIOValues

Parameters

gpioId	GPIO id to be queried.
value_p	Output parameter that will be filled with the GPIO value.

5.17.4.17 CyU3PReturnStatus_t CyU3PGpioSimpleSetValue (uint8_t *gpioId*, CyBool_t *value*)

Update the state of a simple GPIO output pin.

Description

This function updates the state of a simple GPIO output pin. This API works only for simple GPIOs and will be faster than the more generic CyU3PGpioSetValue API call.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the GPIO id is invalid.

See Also

CyU3PGpioSetSimpleConfig
CyU3PGpioSetValue
CyU3PGpioSimpleGetValue

Parameters

gpioId	GPIO id to be modified.
value	Value to set on the GPIO pin.

5.17.4.18 void CyU3PRegisterGpioCallBack (CyU3PGpioIntrCb_t *gpioIntrCb*)

Register a callback for notification of GPIO events.

Description

This function registers a callback function for notification of GPIO related interrupts. The callback can also be registered through the CyU3PGpioInit API call itself.

Return value

None

See Also

CyU3PGpioIntrCb_t
CyU3PGpioInit

Parameters

gpioIntrCb	Callback function pointer.

5.18 firmware/u3p_firmware/inc/cyu3i2c.h File Reference

The I2C driver in the FX3 firmware provides a set of APIs to configure the I2C interface properties and to perform I2C data transfers from/to one or more slave devices.

```
#include <cyu3types.h>
#include <cyu3system.h>
#include <cyu3lpp.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PI2cConfig_t
  Structure defining the I2C interface configuration.
- struct CyU3PI2cPreamble_t
  Structure defining the preamble to be sent on the I2C interface.

Macros

- #define CY_U3P_I2C_DEFAULT_LOCK_TIMEOUT (CYU3P_WAIT_FOREVER)
  Delay duration to wait to get a lock on the I2C block.

Typedefs

- typedef enum CyU3PI2cEvt_t CyU3PI2cEvt_t
  List of I2C related event types.
- typedef enum CyU3PI2cError_t CyU3PI2cError_t
  List of I2C specific error/status codes.
- typedef struct CyU3PI2cConfig_t CyU3PI2cConfig_t
  Structure defining the I2C interface configuration.
- typedef struct CyU3PI2cPreamble_t CyU3PI2cPreamble_t
  Structure defining the preamble to be sent on the I2C interface.
- typedef void(∗ CyU3PI2cIntrCb_t)(CyU3PI2cEvt_t evt, CyU3PI2cError_t error)
  I2C event callback function type.

Enumerations

- 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 }
  List of I2C related event types.
- 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 }
  List of I2C specific error/status codes.

Functions

- CyU3PReturnStatus_t CyU3PI2cInit (void)

  Starts the I2C interface block on the FX3.
- CyU3PReturnStatus_t CyU3PI2cDeInit (void)

  Stops the I2C module.
- CyU3PReturnStatus_t CyU3PI2cSetConfig (CyU3PI2cConfig_t *config, CyU3PI2cIntrCb_t cb)

  Sets the I2C interface parameters.
- CyU3PReturnStatus_t CyU3PI2cSendCommand (CyU3PI2cPreamble_t *preamble, uint32_t byteCount, CyBool_t isRead)

  Initiate a read or write operation to the I2C slave.
- CyU3PReturnStatus_t CyU3PI2cTransmitBytes (CyU3PI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount)

  Writes data to an I2C Slave in register mode.
- CyU3PReturnStatus_t CyU3PI2cReceiveBytes (CyU3PI2cPreamble_t *preamble, uint8_t *data, uint32_t byteCount, uint32_t retryCount)

  Read data from the I2C slave in register mode.
- CyU3PReturnStatus_t CyU3PI2cWaitForAck (CyU3PI2cPreamble_t *preamble, uint32_t retryCount)

  Wait until the I2C slave ACKs the preamble.
- CyU3PReturnStatus_t CyU3PI2cWaitForBlockXfer (CyBool_t isRead)

  Wait until the ongoing I2C data transfer is finished.
- CyU3PReturnStatus_t CyU3PI2cGetErrorCode (CyU3PI2cError_t *error_p)

  Retrieves the error code as defined by CyU3PI2cError_t.
- void CyU3PRegisterI2cCallBack (CyU3PI2cIntrCb_t i2cIntrCb)

  Register a callback function for I2C event notifications.

5.18.1 Detailed Description

The I2C driver in the FX3 firmware provides a set of APIs to configure the I2C interface properties and to perform I2C data transfers from/to one or more slave devices.

5.18.2 Typedef Documentation

5.18.2.1 typedef struct CyU3PI2cConfig_t CyU3PI2cConfig_t

Structure defining the I2C interface configuration.

Description

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

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

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

See Also

CyU3PI2cSetConfig

5.18.2.2 typedef enum CyU3PI2cError_t CyU3PI2cError_t

List of I2C specific error/status codes.

Description

This type lists the various I2C specific error/status codes that are sent to the event callback as event data, when the event type is CY_U3P_I2C_ERROR_EVT.

See Also

CyU3PI2cEvt_t
CyU3PI2cIntrCb_t

5.18.2.3 typedef enum CyU3PI2cEvt_t CyU3PI2cEvt_t

List of I2C related event types.

Description

This enumerated type lists the various I2C related event codes that are sent to the user application through the I2C event callback.

Note

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after the CY_U3P_I2C_EVENT_RX_DONE event is delivered. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

See Also

CyU3PI2cError_t
CyU3PI2cIntrCb_t

5.18.2.4 typedef void(∗ CyU3PI2cIntrCb_t)(CyU3PI2cEvt_t evt,CyU3PI2cError_t error)

I2C event callback function type.

Description

This type defines the signature of the callback functions to be registered to receive I2C related events. I2C event callbacks can only be used in the DMA mode of data transfer.

See Also

CyU3PI2cEvt_t
CyU3PI2cError_t
CyU3PRegisterI2cCallBack

5.18.2.5 typedef struct CyU3PI2cPreamble_t CyU3PI2cPreamble_t

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

Description

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

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

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

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

Typical I2C EEPROM page Read operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

Byte 3:

Bit 7 - 1: Slave address.

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

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

Typical I2C EEPROM page Write operation:

Byte 0:

Bit 7 - 1: Slave address.

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

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

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

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

See Also

CyU3PI2cTransmitBytes
CyU3PI2cReceiveBytes

5.18.3 Enumeration Type Documentation

5.18.3.1 enum CyU3PI2cError_t

List of I2C specific error/status codes.

Description

This type lists the various I2C specific error/status codes that are sent to the event callback as event data, when the event type is CY_U3P_I2C_ERROR_EVT.

See Also

CyU3PI2cEvt_t
CyU3PI2cIntrCb_t

Enumerator

CY_U3P_I2C_ERROR_NAK_BYTE_0 Slave NACK-ed the zeroth byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_1 Slave NACK-ed the first byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_2 Slave NACK-ed the second byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_3 Slave NACK-ed the third byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_4 Slave NACK-ed the fourth byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_5 Slave NACK-ed the fifth byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_6 Slave NACK-ed the sixth byte of the preamble.

CY_U3P_I2C_ERROR_NAK_BYTE_7 Slave NACK-ed the seventh byte of the preamble.

CY_U3P_I2C_ERROR_NAK_DATA Slave sent a NACK during the data phase of a transfer.

CY_U3P_I2C_ERROR_PREAMBLE_EXIT_NACK_ACK Poll operation has exited due to the slave returning an ACK or a NACK handshake.

CY_U3P_I2C_ERROR_PREAMBLE_EXIT Poll operation with address repetition timed out.

CY_U3P_I2C_ERROR_NAK_TX_UNDERFLOW Underflow in buffer during transmit/write operation.

CY_U3P_I2C_ERROR_NAK_TX_OVERFLOW Overflow of buffer during transmit operation.

CY_U3P_I2C_ERROR_NAK_RX_UNDERFLOW Underflow in buffer during receive/read operation.

CY_U3P_I2C_ERROR_NAK_RX_OVERFLOW Overflow of buffer during receive operation.

5.18.3.2 enum CyU3PI2cEvt_t

List of I2C related event types.

Description

This enumerated type lists the various I2C related event codes that are sent to the user application through the I2C event callback.

Note

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after the CY_U3P_I2C_EVENT_RX_DONE event is delivered. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

See Also

CyU3PI2cError_t
CyU3PI2cIntrCb_t

Enumerator

CY_U3P_I2C_EVENT_RX_DONE Reception is completed

CY_U3P_I2C_EVENT_TX_DONE Transmission is done

CY_U3P_I2C_EVENT_TIMEOUT Bus timeout has happened

CY_U3P_I2C_EVENT_LOST_ARBITRATION Lost I2C bus arbitration, probably due to another I2C bus master.

CY_U3P_I2C_EVENT_ERROR I2C transfer has resulted in an error.

5.18.4 Function Documentation

5.18.4.1 CyU3PReturnStatus_t CyU3PI2cDeInit (void)

Stops the I2C module.

Description

This function disables and powers off the I2C interface. This function can be used to shut off the interface, to save power when it is not in use.

Return value

CY_U3P_SUCCESS - If the DeInit is successful.

CY_U3P_ERROR_NOT_STARTED - If the I2C module has not been previously initialized.

See Also

CyU3PI2cInit

5.18.4.2 CyU3PReturnStatus_t CyU3PI2cGetErrorCode (CyU3PI2cError_t * *error_p*)

Retrieves the error code as defined by CyU3PI2cError_t.

Description

This function can be used to retrieve the actual I2C error code once the register mode I2C transfer APIs have returned the CY_U3P_ERROR_FAILURE error code.

Return value

CY_U3P_SUCCESS - If the error code is fetched successfully.

CY_U3P_ERROR_NULL_POINTER - When the error_p pointer passed is NULL.

CY_U3P_ERROR_NOT_STARTED - When there is no error flagged.

CY_U3P_ERROR_MUTEX_FAILURE - Failure to obtain a lock on the I2C block.

See Also

CyU3PI2cTransmitBytes
CyU3PI2cReceiveBytes
CyU3PI2cWaitForAck

Parameters

error_p	Return pointer to be filled with the error code.

5.18.4.3 CyU3PReturnStatus_t CyU3PI2cInit (void)

Starts the I2C interface block on the FX3.

Description

This function powers up the I2C interface block on the FX3 device and is expected to be the first I2C API function that is called by the application. This function also sets up the I2C interface at a default rate of 100KHz.

Return value

CY_U3P_SUCCESS - When the Init is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When I2C has not been enabled in IO configuration.

CY_U3P_ERROR_ALREADY_STARTED - When the I2C has been already initialized.

See Also

CyU3PI2cDeInit
CyU3PI2cSetConfig

5.18.4.4 CyU3PReturnStatus_t CyU3PI2cReceiveBytes (CyU3PI2cPreamble_t * *preamble*, uint8_t * *data*, uint32_t *byteCount*, uint32_t *retryCount*)

Read data from the I2C slave in register mode.

Description

This function is used to read data one byte at a time from an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode. The function call can be repeated when CY_U3P_ERROR_TIMEOUT is returned without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added by the caller.

The API will return when FX3 has received the data. If the slave device requires additional time before responding to other commands, then either sufficient delay must be provided; or the CyU3PI2cWaitForAck API can be used. Refer to the I2C slave datasheet for more details on how to identify when the slave is ready.

This function internally uses the CyU3PI2cSendCommand function.

Return value

CY_U3P_SUCCESS - When the transfer is completed successfully.

CY_U3P_ERROR_NULL_POINTER - If the preamble or data parameter is NULL.

CY_U3P_ERROR_FAILURE - When a transfer fails with an error defined in CyU3PI2cError_t.

CY_U3P_ERROR_BLOCK_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.

CY_U3P_ERROR_NOT_CONFIGURED - If the I2C block is not configured for register mode transfers.

CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.

CY_U3P_ERROR_LOST_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.

CY_U3P_ERROR_MUTEX_FAILURE - Failure to obtain a lock on the I2C block.

See Also

CyU3PI2cPreamble_t
CyU3PI2cSetConfig
CyU3PI2cSendCommand
CyU3PI2cTransmitBytes
CyU3PI2cWaitForAck

Parameters

preamble	Preamble information to be sent out before the data transfer.
data	Pointer to buffer where the data is to be placed.
byteCount	Size of the transfer in bytes.
retryCount	Number of times to retry request in case of a NAK response or error.

5.18.4.5 CyU3PReturnStatus_t CyU3PI2cSendCommand (CyU3PI2cPreamble_t * *preamble*, uint32_t *byteCount*, CyBool_t *isRead*)

Initiate a read or write operation to the I2C slave.

Description

This function is used to send the extended preamble over the I2C bus. This is used in conjunction with data transfer phase in DMA mode. The function is also called from the API library for register mode operation.

The byteCount represents the amount of data to be read or written in the data phase and the isRead parameter specifies the direction of transfer. The transfer will happen through the I2C Consumer / Producer DMA channel if the I2C interface is configured in DMA mode, or through the I2C Ingress/Egress registers if the interface is configured in register mode.

The CyU3PI2cWaitForBlockXfer API or the CY_U3P_I2C_EVENT_RX_DONE or CY_U3P_I2C_EVENT_TX_DONE event callbacks can be used to detect the end of a DMA data transfer that is requested through this function.

Return value

CY_U3P_SUCCESS - When the SendCommand is successful.

CY_U3P_ERROR_NULL_POINTER - When the preamble is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - If the preamble or byteCount arguments are invalid.

CY_U3P_ERROR_TIMEOUT - When the I2C bus is busy.

CY_U3P_ERROR_MUTEX_FAILURE - When there is a failure in acquiring a mutex lock.

See Also

CyU3PI2cPreamble_t
CyU3PI2cSetConfig
CyU3PI2cTransmitBytes
CyU3PI2cReceiveBytes
CyU3PI2cWaitForAck
CyU3PI2cWaitForBlockXfer

Parameters

preamble	Preamble information to be sent out before the data transfer.
byteCount	Size of the transfer in bytes.
isRead	Direction of transfer: CyTrue: Read, CyFalse: Write.

5.18.4.6 CyU3PReturnStatus_t CyU3PI2cSetConfig (CyU3PI2cConfig_t * *config*, CyU3PI2cIntrCb_t *cb*)

Sets the I2C interface parameters.

Description

This function is used to configure the I2C master interface based on the desired baud rate and address length settings to talk to the desired slave.

This function should be called repeatedly to change the settings if different settings are to be used to communicate with different slave devices. This can be called on the fly repetitively without calling CyU3PI2cInit.

In DMA mode, the callback parameter "cb" passed to this function is used to notify the user of data transfer completion or error conditions. In register mode, all APIs are blocking in nature. So in these cases, the callback argument is not relevant and the user must pass NULL as cb when using the register mode.

Bitrate calculation: The maximum bitrate supported is 1 MHz and minimum bitrate is 100 KHz. It should be noted that even though the dividers and the API allows frequencies above and below the rated range, the device behaviour is not guaranteed.

The I2C block on the FX3 needs to be clocked at 10X the desired bit rate. As the I2C block clock is derived from the FX3 SYSTEM clock, the actual bit rate will not be the exact programmed value. As the hardware only supports clock division by integer and half dividers, the firmware uses the closest approximation possible.

Return value

CY_U3P_SUCCESS - When the SetConfig is successful.

CY_U3P_ERROR_NOT_STARTED - When the I2C has not been initialized.

CY_U3P_ERROR_NULL_POINTER - When the config parameter is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - When the arguments are incorrect.

CY_U3P_ERROR_TIMEOUT - When there is timeout happening during configuration.

CY_U3P_ERROR_MUTEX_FAILURE- When there is a failure in acquiring a mutex lock.

See Also

CyU3PI2cIntrCb_t CyU3PI2cConfig_t CyU3PI2cInit CyU3PI2cSendCommand CyU3PI2cTransmitBytes CyU3PI2cReceiveBytes

Parameters

config	I2C configuration settings
cb	Callback for getting the events

5.18.4.7 CyU3PReturnStatus_t CyU3PI2cTransmitBytes (CyU3PI2cPreamble_t * *preamble*, uint8_t * *data*, uint32_t *byteCount*, uint32_t *retryCount*)

Writes data to an I2C Slave in register mode.

Description

This function is used to write data one byte at a time to an I2C slave. This function requires that the I2C interface be configured in register (non-DMA) mode. The function call can be repeated when CY_U3P_ERROR_TIMEOUT is returned without any error recovery. The retry is done continuously without any delay. If any delay is required, then it should be added by the caller.

The API will return when FX3 has transmitted the data. If the slave device requires additional time for completing the write operation, then either sufficient delay must be provided; or the CyU3PI2cWaitForAck API can be used. Refer to the I2C slave datasheet for more details on how to identify when the write is complete.

This function internally uses the CyU3PI2cSendCommand function.

Return value

CY_U3P_SUCCESS - When the TransmitBytes is successful.

CY_U3P_ERROR_NULL_POINTER - If the preamble or data parameter is NULL.

CY_U3P_ERROR_FAILURE - When a transfer fails with an error defined in CyU3PI2cError_t.

CY_U3P_ERROR_BLOCK_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.

CY_U3P_ERROR_NOT_CONFIGURED - If the I2C block is not configured for register mode transfers.

CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.

CY_U3P_ERROR_LOST_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.

CY_U3P_ERROR_MUTEX_FAILURE - Failure to obtain a lock on the I2C block.

See Also

CyU3PI2cPreamble_t
CyU3PI2cSetConfig
CyU3PI2cSendCommand
CyU3PI2cReceiveBytes
CyU3PI2cWaitForAck

Parameters

preamble	Preamble information to be sent out before the data transfer.
data	Pointer to buffer containing data to be written.
byteCount	Size of the transfer in bytes.
retryCount	Number of times to retry request in case of a slave NAK response.

5.18.4.8 CyU3PReturnStatus_t CyU3PI2cWaitForAck (CyU3PI2cPreamble_t * *preamble*, uint32_t *retryCount*)

Wait until the I2C slave ACKs the preamble.

Description

This function waits for a ACK handshake from the slave, and can be used to ensure that the slave device has reached a desired state before issuing the next transaction or shutting the interface down.

The API repeats the provided preamble bytes continuously until all bytes of the preamble are ACKed, or the specified retryCount has been reached.

Return value

CY_U3P_SUCCESS - When the device returns the desired handshake.

CY_U3P_ERROR_NULL_POINTER - If the preamble is NULL.

CY_U3P_ERROR_FAILURE - When a preamble transfer fails with error defined by CyU3PI2cError_t.

CY_U3P_ERROR_NOT_CONFIGURED - When the I2C is not initialized or not configured.

CY_U3P_ERROR_BLOCK_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.

CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.

CY_U3P_ERROR_LOST_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.

CY_U3P_ERROR_MUTEX_FAILURE - Failure to obtain a lock on the I2C block.

See Also

CyU3PI2cPreamble_t
CyU3PI2cSendCommand
CyU3PI2cTransmitBytes
CyU3PI2cReceiveBytes

Parameters

preamble	Preamble information to be sent out before the data transfer.
retryCount	Number of times to retry request if the slave continues to NAK.

5.18.4.9 CyU3PReturnStatus_t CyU3PI2cWaitForBlockXfer (CyBool_t *isRead*)

Wait until the ongoing I2C data transfer is finished.

Description

This function waits until the ongoing DMA based I2C transaction is complete. The function returns when FX3 has finished transferring the requested amount of data.

Note

In the case of a DMA read of data that does not fill the DMA buffer(s) associated with the read DMA channel, the DMA transfer remains pending after this API call returns. The data can only be retrieved from the DMA buffer after the DMA transfer is terminated through the CyU3PDmaChannelSetWrapUp API.

Return value

CY_U3P_SUCCESS - When the data transfer has been completed successfully.

CY_U3P_ERROR_NOT_CONFIGURED - When the I2C is not initialized or not configured.

CY_U3P_ERROR_NOT_SUPPORTED - If a I2C callback function has been registered.

CY_U3P_ERROR_FAILURE - When there is a failure defined by CyU3PI2cError_t.

CY_U3P_ERROR_BLOCK_FAILURE - When the I2C block encounters a fatal error and requires to be re-initialized.

CY_U3P_ERROR_TIMEOUT - I2C bus timeout occurred.

CY_U3P_ERROR_LOST_ARBITRATION - Failure due to I2C arbitration error or invalid bus activity.

CY_U3P_ERROR_MUTEX_FAILURE - Failure to obtain a lock on the I2C block.

See Also

CyU3PI2cSendCommand

Parameters

isRead	Type of operation to wait on: CyTrue: Read, CyFalse: Write

5.18.4.10 void CyU3PRegisterI2cCallBack (CyU3PI2cIntrCb_t *i2cIntrCb*)

Register a callback function for I2C event notifications.

Description

This function registers a callback function that will be called for notification of I2C interrupts.

Return value

None

See Also

CyU3PI2cIntrCb_t

Parameters

i2cIntrCb	Callback function pointer.

5.19 firmware/u3p_firmware/inc/cyu3i2s.h File Reference

The I2S (Inter-IC Sound) interface is a serial interface defined for communication of stereophonic audio data between devices. The FX3 device includes a I2S master interface which can be connected to an I2S peripheral. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PI2sConfig_t

  I2S interface configuration parameters.

Macros

- #define CY_U3P_I2S_DEFAULT_LOCK_TIMEOUT (CYU3P_WAIT_FOREVER)

  Delay duration to wait to get a lock on the I2S block.

Typedefs

- typedef enum CyU3PI2sEvt_t CyU3PI2sEvt_t

  List of I2S related event types.
- typedef enum CyU3PI2sError_t CyU3PI2sError_t

  List of I2S specific error/status codes.
- typedef enum CyU3PI2sSampleWidth_t CyU3PI2sSampleWidth_t

  List of supported bit widths for the I2S interface.
- typedef enum CyU3PI2sSampleRate_t CyU3PI2sSampleRate_t

  List of supported sample rates.
- typedef enum CyU3PI2sPadMode_t CyU3PI2sPadMode_t

  List of the supported padding modes.
- typedef struct CyU3PI2sConfig_t CyU3PI2sConfig_t

  I2S interface configuration parameters.
- typedef void(∗ CyU3PI2sIntrCb_t)(CyU3PI2sEvt_t evt, CyU3PI2sError_t error)

  Prototype of I2S event callback function.

Enumerations

- enum CyU3PI2sEvt_t { CY_U3P_I2S_EVENT_TXL_DONE = 0, CY_U3P_I2S_EVENT_TXR_DONE, CY_U3P_I2S_EVENT_PAUSED, CY_U3P_I2S_EVENT_ERROR }

  List of I2S related event types.
- enum CyU3PI2sError_t { CY_U3P_I2S_ERROR_LTX_UNDERFLOW = 11, CY_U3P_I2S_ERROR_RTX_UNDERFLOW, CY_U3P_I2S_ERROR_LTX_OVERFLOW, CY_U3P_I2S_ERROR_RTX_OVERFLOW }

  List of I2S specific error/status codes.
- enum CyU3PI2sSampleWidth_t {
  CY_U3P_I2S_WIDTH_8_BIT = 0, CY_U3P_I2S_WIDTH_16_BIT, CY_U3P_I2S_WIDTH_18_BIT, CY_U3P_I2S_WIDTH_24_BIT,
  CY_U3P_I2S_WIDTH_32_BIT, CY_U3P_I2S_NUM_BIT_WIDTH }

  List of supported bit widths for the I2S interface.
- enum CyU3PI2sSampleRate_t {
  CY_U3P_I2S_SAMPLE_RATE_8KHz = 8000, CY_U3P_I2S_SAMPLE_RATE_16KHz = 16000, CY_U3P_I2S_SAMPLE_RATE_32KHz = 32000, CY_U3P_I2S_SAMPLE_RATE_44_1KHz = 44100,
  CY_U3P_I2S_SAMPLE_RATE_48KHz = 48000, CY_U3P_I2S_SAMPLE_RATE_96KHz = 96000, CY_U3P_I2S_SAMPLE_RATE_192KHz = 192000 }

  List of supported sample rates.
- enum CyU3PI2sPadMode_t {
  CY_U3P_I2S_PAD_MODE_NORMAL = 0, CY_U3P_I2S_PAD_MODE_LEFT_JUSTIFIED, CY_U3P_I2S_PAD_MODE_RIGHT_JUSTIFIED, CY_U3P_I2S_PAD_MODE_RESERVED,
  CY_U3P_I2S_PAD_MODE_CONTINUOUS, CY_U2P_I2S_NUM_PAD_MODES }

  List of the supported padding modes.

Functions

- CyU3PReturnStatus_t CyU3PI2sInit (void)

  Starts the I2S interface block on the FX3.
- CyU3PReturnStatus_t CyU3PI2sDeInit (void)

  Stops the I2S interface block on the FX3.
- CyU3PReturnStatus_t CyU3PI2sSetConfig (CyU3PI2sConfig_t ∗config, CyU3PI2sIntrCb_t cb)

  Sets the I2S interface parameters.
- CyU3PReturnStatus_t CyU3PI2sTransmitBytes (uint8_t ∗lData, uint8_t ∗rData, uint8_t lByteCount, uint8_t rByteCount)

Transmits data byte by byte over the I2S interface.

- CyU3PReturnStatus_t CyU3PI2sSetMute (CyBool_t isMute)

  Mute or unmute the I2S output stream.

- CyU3PReturnStatus_t CyU3PI2sSetPause (CyBool_t isPause)

  Pause or resume I2S data transfers.

- void CyU3PRegisterI2sCallBack (CyU3PI2sIntrCb_t i2sIntrCb)

  This function registers a callback function for notification of I2S interrupts.

5.19.1 Detailed Description

The I2S (Inter-IC Sound) interface is a serial interface defined for communication of stereophonic audio data between devices. The FX3 device includes a I2S master interface which can be connected to an I2S peripheral. The I2S driver module provides functions to configure the I2S interface and to send mono or stereo audio output on the I2S link.

5.19.2 Typedef Documentation

5.19.2.1 typedef struct CyU3PI2sConfig_t CyU3PI2sConfig_t

I2S interface configuration parameters.

Description

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

See Also

CyU3PI2sSetConfig

5.19.2.2 typedef enum CyU3PI2sError_t CyU3PI2sError_t

List of I2S specific error/status codes.

Description

This type lists the various I2S specific error/status codes that are sent to the event callback as event data, when the event type is CY_U3P_I2S_ERROR_EVT.

See Also

CyU3PI2sEvt_t
CyU3PI2sIntrCb_t

5.19.2.3 typedef enum CyU3PI2sEvt_t CyU3PI2sEvt_t

List of I2S related event types.

Description

This enumeration lists the various I2S related event codes that are notified to the user application through an event callback.

See Also

CyU3PI2sIntrCb_t
CyU3PI2sError_t

5.19.2.4 typedef void(* CyU3PI2sIntrCb_t)(CyU3PI2sEvt_t evt,CyU3PI2sError_t error)

Prototype of I2S event callback function.

Description

This function type defines a callback to be called when an I2S interrupt has been received. A function of this type can be registered with the I2S driver as a callback function and will be called whenever an event of interest occurs.

See Also

CyU3PI2sEvt_t
CyU3PI2sError_t
CyU3PRegisterI2sCallBack

5.19.2.5 typedef enum CyU3PI2sPadMode_t CyU3PI2sPadMode_t

List of the supported padding modes.

Description

This types lists the padding modes supported on the I2S interface.

See Also

CyU3PI2sConfig_t

5.19.2.6 typedef enum CyU3PI2sSampleRate_t CyU3PI2sSampleRate_t

List of supported sample rates.

Description

This type lists the supported sample rates for audio playback through the I2S interface.

See Also

CyU3PI2sConfig_t

5.19.2.7 typedef enum CyU3PI2sSampleWidth_t CyU3PI2sSampleWidth_t

List of supported bit widths for the I2S interface.

Description

This type lists the supported bit-widths on the I2S interface.

See Also

CyU3PI2sConfig_t

5.19.3 Enumeration Type Documentation

5.19.3.1 enum CyU3PI2sError_t

List of I2S specific error/status codes.

Description

This type lists the various I2S specific error/status codes that are sent to the event callback as event data, when the event type is CY_U3P_I2S_ERROR_EVT.

See Also

CyU3PI2sEvt_t
CyU3PI2sIntrCb_t

Enumerator

CY_U3P_I2S_ERROR_LTX_UNDERFLOW A left channel underflow occurred.

CY_U3P_I2S_ERROR_RTX_UNDERFLOW A right channel underflow occurred.

CY_U3P_I2S_ERROR_LTX_OVERFLOW A left channel overflow occurred.

CY_U3P_I2S_ERROR_RTX_OVERFLOW A right channel overflow occurred.

5.19.3.2 enum CyU3PI2sEvt_t

List of I2S related event types.

Description

This enumeration lists the various I2S related event codes that are notified to the user application through an event callback.

See Also

CyU3PI2sIntrCb_t
CyU3PI2sError_t

Enumerator

CY_U3P_I2S_EVENT_TXL_DONE Transmission of left channel data is complete.

CY_U3P_I2S_EVENT_TXR_DONE Transmission of right channel data is complete.

CY_U3P_I2S_EVENT_PAUSED Pause has taken effect.

CY_U3P_I2S_EVENT_ERROR An I2S error has been detected.

5.19.3.3 enum CyU3PI2sPadMode_t

List of the supported padding modes.

Description

This types lists the padding modes supported on the I2S interface.

See Also

CyU3PI2sConfig_t

Enumerator

CY_U3P_I2S_PAD_MODE_NORMAL I2S normal operation.

CY_U3P_I2S_PAD_MODE_LEFT_JUSTIFIED Left justified.

CY_U3P_I2S_PAD_MODE_RIGHT_JUSTIFIED Right justified.

CY_U3P_I2S_PAD_MODE_RESERVED Reserved mode.

CY_U3P_I2S_PAD_MODE_CONTINUOUS Without padding.

CY_U2P_I2S_NUM_PAD_MODES Number of pad modes.

5.19.3.4 enum CyU3PI2sSampleRate_t

List of supported sample rates.

Description

This type lists the supported sample rates for audio playback through the I2S interface.

See Also

CyU3PI2sConfig_t

Enumerator

CY_U3P_I2S_SAMPLE_RATE_8KHz 8 KHz

CY_U3P_I2S_SAMPLE_RATE_16KHz 16 KHz

CY_U3P_I2S_SAMPLE_RATE_32KHz 32 KHz

CY_U3P_I2S_SAMPLE_RATE_44_1KHz 44.1 KHz

CY_U3P_I2S_SAMPLE_RATE_48KHz 48 KHz

CY_U3P_I2S_SAMPLE_RATE_96KHz 96 KHz

CY_U3P_I2S_SAMPLE_RATE_192KHz 192 KHz

5.19.3.5 enum CyU3PI2sSampleWidth_t

List of supported bit widths for the I2S interface.

Description

This type lists the supported bit-widths on the I2S interface.

See Also

CyU3PI2sConfig_t

Enumerator

CY_U3P_I2S_WIDTH_8_BIT 8 bit

CY_U3P_I2S_WIDTH_16_BIT 16 bit

CY_U3P_I2S_WIDTH_18_BIT 18 bit

CY_U3P_I2S_WIDTH_24_BIT 24 bit

CY_U3P_I2S_WIDTH_32_BIT 32 bit

CY_U3P_I2S_NUM_BIT_WIDTH Number of options.

5.19.4 Function Documentation

5.19.4.1 CyU3PReturnStatus_t CyU3PI2sDeInit (void)

Stops the I2S interface block on the FX3.

Description

This function disables and powers off the I2S interface. This function can be used to shut off the interface to save power when it is not in use.

Return value

CY_U3P_SUCCESS - When the de-init is successful.

CY_U3P_ERROR_NOT_STARTED - When the module has not been previously initialized.

See Also

CyU3PI2sInit

5.19.4.2 CyU3PReturnStatus_t CyU3PI2sInit (void)

Starts the I2S interface block on the FX3.

Description

This function powers up the I2S interface block on the FX3 device, and is expected to be the first I2S API function that is called by the application.

This function sets up the clock to a default value of CY_U3P_I2S_SAMPLE_RATE_8KHz.

Return value

CY_U3P_SUCCESS - When the init is successful.

CY_U3P_ERROR_ALREADY_STARTED - When the I2S block has already been initialized.

CY_U3P_ERROR_NOT_CONFIGURED - When the I2S block has not been enabled in the IOMatrix.

See Also

CyU3PI2sDeinit
CyU3PI2sSetConfig
CyU3PI2sTransmitBytes
CyU3PI2sSetMute
CyU3PI2sSetPause

5.19.4.3 CyU3PReturnStatus_t CyU3PI2sSetConfig (CyU3PI2sConfig_t * *config*, CyU3PI2sIntrCb_t *cb*)

Sets the I2S interface parameters.

Description

This function is used to configure the I2S master interface based on the desired settings. This function should be called repeatedly every time the output stream parameters change.

The callback parameter is used to specify an event callback function that will be called by the driver when an I2S interrupt occurs.

Return value

CY_U3P_SUCCESS - When the SetConfig is successful.

CY_U3P_ERROR_NOT_STARTED - When the I2S has not been initialized.

CY_U3P_ERROR_NULL_POINTER - When the config pointer is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - When the configuration parameters are incorrect.

CY_U3P_ERROR_TIMEOUT - If there is a timeout while trying to clear the left and right channel pipes.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the I2S block.

See Also

CyU3PI2sConfig_t
CyU3PI2sIntrCb_t
CyU3PI2sInit
CyU3PI2sTransmitBytes
CyU3PI2sSetMute
CyU3PI2sSetPause

Parameters

config	I2S configuration settings.
cb	Callback for getting the events.

5.19.4.4 CyU3PReturnStatus_t CyU3PI2sSetMute (CyBool_t *isMute*)

Mute or unmute the I2S output stream.

Description

This function sets the I2C master to mute or unmute the output stream. When configured for mute, the I2S master drives zero samples instead of actual data output.

Return value

CY_U3P_SUCCESS - When the mute control is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When the I2S interface has not been configured.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the I2S block.

See Also

CyU3PI2sInit
CyU3PI2sSetConfig

Parameters

isMute	CyTrue: Mute the I2S, CyFalse: Un-mute the I2S.

5.19.4.5 CyU3PReturnStatus_t CyU3PI2sSetPause (CyBool_t *isPause*)

Pause or resume I2S data transfers.

Description

This function is used to pause or resume the data transfer on the I2S interface.

Return value

CY_U3P_SUCCESS - When the pause control is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When the I2S interface has not been configured.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the I2S block.

See Also

CyU3PI2sInit
CyU3PI2sSetConfig

Parameters

isPause	CyTrue: pause the I2S, CyFalse: resume the I2S.

5.19.4.6 CyU3PReturnStatus_t CyU3PI2sTransmitBytes (uint8_t * *lData*, uint8_t * *rData*, uint8_t *lByteCount*, uint8_t *rByteCount*)

Transmits data byte by byte over the I2S interface.

Description

This function sends the data over the I2S interface in register mode. This is allowed only when the I2S interface block is configured for register mode transfer. The data involved in this transfer is always left justified.

Return value

CY_U3P_SUCCESS - When the data transfer is successful.

CY_U3P_ERROR_NULL_POINTER - When the data pointers are NULL.

CY_U3P_ERROR_TIMEOUT - When a timeout occurs.

CY_U3P_ERROR_NOT_CONFIGURED - If the I2S has not been configured for register mode transfers.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the I2S block.

See Also

CyU3PI2sSetConfig
CyU3PI2sSetMute
CyU3PI2sSetPause

Parameters

lData	Buffer containing data to be sent on the left channel.
rData	Buffer containing data to be sent on the right channel.
lByteCount	Number of bytes to be transferred on the left channel.
rByteCount	Number of bytes to be transferred on the right channel.

5.19.4.7 void CyU3PRegisterI2sCallBack (CyU3PI2sIntrCb_t *i2sIntrCb*)

This function registers a callback function for notification of I2S interrupts.

Description

This function registers a callback function that will be called for notification of I2S interrupts. This can also be done through the CyU3PI2sInit API call.

Return value

None

See Also

CyU3PI2sIntrCb_t
CyU3PI2sInit

Parameters

i2sIntrCb	Callback function pointer.

5.20 firmware/u3p_firmware/inc/cyu3lpp.h File Reference

All of the serial peripherals and GPIOs on the FX3 device share a set of common hardware resources that need to be initialized before any of these interfaces can be used. Further, the drivers for all of the serial peripherals share a single RTOS thread; because the CPU load due to each of these is expected to be minimal. This file defines the data types and APIs that are used for the central management of all these serial peripheral blocks.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3system.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PGpioClock_t

  Clock configuration information for the GPIO block.

Macros

- #define **CY_U3P_LPP_EVENT_GPIO_INTR** (1 << 3)
- #define **CY_U3P_LPP_EVENT_I2S_INTR** (1 << 4)
- #define **CY_U3P_LPP_EVENT_I2C_INTR** (1 << 5)
- #define **CY_U3P_LPP_EVENT_UART_INTR** (1 << 6)
- #define **CY_U3P_LPP_EVENT_SPI_INTR** (1 << 7)

Typedefs

- typedef enum CyU3PGpioIoMode_t CyU3PGpioIoMode_t

  List of IO modes.
- typedef enum CyU3PGpioSimpleClkDiv_t CyU3PGpioSimpleClkDiv_t

  Clock divider values for sampling simple GPIOs.
- typedef struct CyU3PGpioClock_t CyU3PGpioClock_t

  Clock configuration information for the GPIO block.
- typedef void(* CyU3PLppInterruptHandler)(void)

  Serial peripheral interrupt handler function type.

Enumerations

- enum CyU3PGpioIoMode_t { CY_U3P_GPIO_IO_MODE_NONE = 0, CY_U3P_GPIO_IO_MODE_WPU, CY_U3P_GPIO_IO_MODE_WPD }

  List of IO modes.
- enum CyU3PGpioSimpleClkDiv_t { CY_U3P_GPIO_SIMPLE_DIV_BY_2 = 0, CY_U3P_GPIO_SIMPLE_DIV_BY_4, CY_U3P_GPIO_SIMPLE_DIV_BY_16, CY_U3P_GPIO_SIMPLE_DIV_BY_64, CY_U3P_GPIO_SIMPLE_NUM_DIV }

  Clock divider values for sampling simple GPIOs.

Functions

- CyBool_t CyU3PLppGpioBlockIsOn (void)

  Check if the boot firmware has left the GPIO block powered ON.
- CyU3PReturnStatus_t CyU3PLppInit (CyU3PLppModule_t lppModule, CyU3PLppInterruptHandler intrHandler)

  Register the specified peripheral block as active.
- CyU3PReturnStatus_t CyU3PLppDeInit (CyU3PLppModule_t lppModule)

Register that a specified peripheral block has been made inactive.

- CyU3PReturnStatus_t CyU3PGpioSetClock (CyU3PGpioClock_t *clk_p)

  Enable the GPIO block clocks on the FX3 device.
- CyU3PReturnStatus_t CyU3PI2sSetClock (uint32_t clkRate)

  Set the frequency for and enable the I2S clock.
- CyU3PReturnStatus_t CyU3PI2cSetClock (uint32_t bitRate)

  Set the frequency for and enable the I2C clock.
- CyU3PReturnStatus_t CyU3PUartSetClock (uint32_t baudRate)

  Set the frequency for and enable the UART block.
- CyU3PReturnStatus_t CyU3PSpiSetClock (uint32_t clock)

  Set the frequency for and enable the SPI block.
- CyU3PReturnStatus_t CyU3PSpiStopClock (void)

  Disable the SPI block clock.
- CyU3PReturnStatus_t CyU3PI2cStopClock (void)

  Disable the I2C block clock.
- CyU3PReturnStatus_t CyU3PGpioStopClock (void)

  Disable the GPIO block clocks.
- CyU3PReturnStatus_t CyU3PI2sStopClock (void)

  Disable the I2S block clock.
- CyU3PReturnStatus_t CyU3PUartStopClock (void)

  Disable the UART block clock.
- CyU3PReturnStatus_t CyU3PSetI2cDriveStrength (CyU3PDriveStrengthState_t i2cDriveStrength)

  Set the IO drive strength for the I2C interface.
- CyU3PReturnStatus_t CyU3PGpioSetIoMode (uint8_t gpioId, CyU3PGpioIoMode_t ioMode)

  Set IO mode for the selected GPIO.
- CyU3PReturnStatus_t CyU3PSetGpioDriveStrength (CyU3PDriveStrengthState_t gpioDriveStrength)

  Set the IO drive strength for all FX3 GPIOs.

5.20.1 Detailed Description

All of the serial peripherals and GPIOs on the FX3 device share a set of common hardware resources that need to be initialized before any of these interfaces can be used. Further, the drivers for all of the serial peripherals share a single RTOS thread; because the CPU load due to each of these is expected to be minimal. This file defines the data types and APIs that are used for the central management of all these serial peripheral blocks.

5.20.2 Typedef Documentation

5.20.2.1 typedef struct CyU3PGpioClock_t CyU3PGpioClock_t

Clock configuration information for the GPIO block.

Description

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

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

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

See Also

CyU3PGpioSetClock
CyU3PSysClockSrc_t
CyU3PGpioSimpleClkDiv_t

5.20.2.2 typedef enum CyU3PGpioIoMode_t CyU3PGpioIoMode_t

List of IO modes.

Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

See Also

CyU3PGpioSetIoMode

5.20.2.3 typedef enum CyU3PGpioSimpleClkDiv_t CyU3PGpioSimpleClkDiv_t

Clock divider values for sampling simple GPIOs.

Description

Sampling and updates of simple GPIOs on the FX3 device are performed based on a clock that is derived from the GPIO fast clock. This type lists the possible divisor values that can be used to derive this clock from the fast clock.

See Also

CyU3PGpioClock_t
CyU3PGpioSetClock

5.20.2.4 typedef void(* CyU3PLppInterruptHandler)(void)

Serial peripheral interrupt handler function type.

Description

Each serial peripheral (I2C, I2S, SPI, UART and GPIO) on the FX3 device can have interrupts associated with it. The drivers for these blocks can register an interrupt handler function that the FX3 firmware framework will call when an interrupt is received. This block specific interrupt handler is registered through the CyU3PLppInit function.

See Also

CyU3PLppInit

5.20.3 Enumeration Type Documentation

5.20.3.1 enum CyU3PGpioIoMode_t

List of IO modes.

Description

The FX3 device supports internal weak pull-ups or pull-downs on its GPIOs. These terminations can be used even on signals that are not used as GPIOs.

This type lists the possible internal IO modes that can be selected for a FX3 GPIO.

See Also

CyU3PGpioSetIoMode

Enumerator

CY_U3P_GPIO_IO_MODE_NONE No internal pull-up or pull-down. Default condition.

CY_U3P_GPIO_IO_MODE_WPU A weak pull-up is provided on the IO.

CY_U3P_GPIO_IO_MODE_WPD A weak pull-down is provided on the IO.

5.20.3.2 enum CyU3PGpioSimpleClkDiv_t

Clock divider values for sampling simple GPIOs.

Description

Sampling and updates of simple GPIOs on the FX3 device are performed based on a clock that is derived from the GPIO fast clock. This type lists the possible divisor values that can be used to derive this clock from the fast clock.

See Also

CyU3PGpioClock_t
CyU3PGpioSetClock

Enumerator

CY_U3P_GPIO_SIMPLE_DIV_BY_2 Simple GPIO clock frequency is fast clock by 2.

CY_U3P_GPIO_SIMPLE_DIV_BY_4 Simple GPIO clock frequency is fast clock by 4.

CY_U3P_GPIO_SIMPLE_DIV_BY_16 Simple GPIO clock frequency is fast clock by 16.

CY_U3P_GPIO_SIMPLE_DIV_BY_64 Simple GPIO clock frequency is fast clock by 64.

CY_U3P_GPIO_SIMPLE_NUM_DIV Number of divider enumerations.

5.20.4 Function Documentation

5.20.4.1 CyU3PReturnStatus_t CyU3PGpioSetClock (CyU3PGpioClock_t * *clk_p*)

Enable the GPIO block clocks on the FX3 device.

Description

The GPIO block on the FX3 device makes use of multiple clocks. This function is used to select the frequency for these clocks and to turn them on.

Return value

CY_U3P_SUCCESS - If the clocks were successfully configured and enabled.

CY_U3P_ERROR_BAD_ARGUMENT - If the clock configuration specified is invalid.

See Also

CyU3PGpioClock_t
CyU3PGpioStopClock

Parameters

clk_p	The desired clock parameters.

5.20.4.2 CyU3PReturnStatus_t CyU3PGpioSetIoMode (uint8_t *gpioId*, CyU3PGpioIoMode_t *ioMode*)

Set IO mode for the selected GPIO.

Description

This function enables or disables internal pull-ups/pull-downs on the selected FX3 IO pin. This IO mode change will take about 5 us to become active.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT - If the gpioId or ioMode requested is invalid.

See Also

CyU3PGpioIoMode_t

Parameters

gpioId	GPIO Pin to be updated.
ioMode	Desired pull-up/pull-down mode.

5.20.4.3 CyU3PReturnStatus_t CyU3PGpioStopClock (void)

Disable the GPIO block clocks.

Description

This function disables all of the clocks associated with the GPIO block. This should only be called after the GPIO block has been de-initialized.

Return value

CY_U3P_SUCCESS - If the GPIO clocks were successfully turned off.

See Also

CyU3PGpioSetClock

5.20.4.4 CyU3PReturnStatus_t CyU3PI2cSetClock (uint32_t *bitRate*)

Set the frequency for and enable the I2C clock.

Description

The clock frequency for the I2C block depends on the desired output bit rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the I2C block needs to run at 10X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS_CLK.

This function sets the internal clock for the I2C block to the value that provides the closest approximation of the desired output bit rate.

Return value

CY_U3P_SUCCESS - If the I2C interface clock was setup as required.

CY_U3P_ERROR_BAD_ARGUMENT - When invalid arqument is passed to the function.

See Also

CyU3PI2cStopClock

Parameters

bitRate	Desired interface clock frequency.

5.20.4.5 CyU3PReturnStatus_t CyU3PI2cStopClock (void)

Disable the I2C block clock.

Description

This function disables the I2C block clock. This should only be called after the I2C block has been de-initialized.

Return value

CY_U3P_SUCCESS - If the I2C clock was successfully turned off.

See Also

CyU3PI2cSetClock

5.20.4.6 CyU3PReturnStatus_t CyU3PI2sSetClock (uint32_t *clkRate*)

Set the frequency for and enable the I2S clock.

Description

The I2S block on the FX3 device needs to be clocked at different rates based on the output audio sample rate. This function sets the I2C block clock frequency and enables the clock.

The internal clock for the I2S block needs to run at 16X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS_CLK.

This function sets the internal clock for the I2S block to the value that provides the closest approximation of the desired output clock frequency.

Return value

CY_U3P_SUCCESS - If the I2S interface clock was setup as required.

CY_U3P_ERROR_BAD_ARGUMENT - When invalid arqument is passed to the function.

See Also

CyU3PI2sStopClock

Parameters

clkRate	Desired interface clock frequency.

5.20.4.7 CyU3PReturnStatus_t CyU3PI2sStopClock (void)

Disable the I2S block clock.

Description

This function disables the I2S block clock. This should only be called after the I2S block has been de-initialized.

Return value

CY_U3P_SUCCESS - If the I2S clock is successfully turned off.

See Also

CyU3PI2sSetClock

5.20.4.8 CyU3PReturnStatus_t CyU3PLppDeInit (CyU3PLppModule_t *lppModule*)

Register that a specified peripheral block has been made inactive.

Description

This function registers that a specified peripheral block is no longer in use. The function along with CyU3PLppInit() keeps track of the peripheral blocks that are ON; and manages the power on/off of the shared resources for these blocks.

The clock for the block being disabled should be turned off only after this function is called.

Return value

CY_U3P_SUCCESS - If the call is successful.

CY_U3P_ERROR_NOT_STARTED - If the serial peripheral being stopped has not been started.

See Also

CyU3PLppInit

Parameters

lppModule	The peripheral block being de-initialized.

5.20.4.9 CyBool_t CyU3PLppGpioBlockIsOn (void)

Check if the boot firmware has left the GPIO block powered ON.

Description

In systems which make use of the boot firmware, it is possible that some of the GPIOs have been left configured by the boot firmware. In such a case, the GPIO block on the FX3 device should not be reset during firmware initialization. This function is used to check whether the boot firmware has requested the GPIO block to be left powered ON across firmware initialization.

Note

Please note that all the pins that have been left configured by the boot firmware need to be selected as simple GPIOs during IO Matrix configuration. Otherwise, these pins will be tri-state because the pin functionality is overridden.

Return value

CyTrue if the boot firmware has left GPIO on.

CyFalse if the GPIO block is turned off.

See Also

CyU3PDeviceConfigureIOMatrix

5.20.4.10 CyU3PReturnStatus_t CyU3PLppInit (CyU3PLppModule_t *lppModule*, CyU3PLppInterruptHandler *intrHandler*)

Register the specified peripheral block as active.

Description

The serial peripheral blocks on the FX3 device share some resources. While the individual peripheral blocks (GPIO, I2C, UART, SPI and I2S) can be turned on/off at runtime; the shared resources need to be kept initialized while any of these blocks are on. This function is used to manage the shared peripheral resources and also to keep track of which peripheral blocks are on.

This function need to be called after initializing the clock for the corresponding peripheral interface.

Return value

CY_U3P_SUCCESS - If the call is successful.

CY_U3P_ERROR_ALREADY_STARTED - The block is already initialized.

CY_U3P_ERROR_INVALID_SEQUENCE - If the block init is being called without turning on the corresponding clock.

See Also

CyU3PLppDeInit
CyU3PUartSetClock
CyU3PI2cSetClock
CyU3PI2sSetClock
CyU3PSpiSetClock

Parameters

lppModule	The peripheral block being initialized.
intrHandler	Interrupt handler function for the peripheral block.

5.20.4.11 CyU3PReturnStatus_t CyU3PSetGpioDriveStrength (CyU3PDriveStrengthState_t *gpioDriveStrength*)

Set the IO drive strength for all FX3 GPIOs.

Description

This function updates the drive strength of all FX3 GPIOs. This is an alternative for setting the drive strength for each port separately.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the GPIO block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the drive strength requested is invalid.

See Also

CyU3PDriveStrengthState_t

Parameters

gpioDrive-Strength	Desired GPIO Drive strength

5.20.4.12 CyU3PReturnStatus_t CyU3PSetI2cDriveStrength (CyU3PDriveStrengthState_t *i2cDriveStrength*)

Set the IO drive strength for the I2C interface.

Description

The function sets the IO Drive strength for the I2C interface signals. The default IO drive strength for I2C is set to CY_U3P_DS_THREE_QUARTER_STRENGTH.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the I2C block has not been initialized.

CY_U3P_ERROR_BAD_ARGUMENT - If the drive strength requested is invalid.

See Also

CyU3PDriveStrengthState_t

Parameters

i2cDriveStrength	Drive strength desired for I2C signals.

5.20.4.13 CyU3PReturnStatus_t CyU3PSpiSetClock (uint32_t *clock*)

Set the frequency for and enable the SPI block.

Description

The clock frequency for the SPI block depends on the desired output bit rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the SPI block needs to run at 2X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS_CLK.

This function sets the internal clock for the SPI block to the value that provides the closest approximation of the desired output bit rate.

Return value

CY_U3P_SUCCESS - If the SPI clock has been successfully turned on.

CY_U3P_ERROR_BAD_ARGUMENT - When invalid arqument is passed to the function.

See Also

CyU3PSpiStopClock

Parameters

clock	Desired output bit rate.

5.20.4.14 CyU3PReturnStatus_t CyU3PSpiStopClock (void)

Disable the SPI block clock.

Description

This function disables the SPI block clock. This should only be called after the SPI block has been de-initialized.

Return value

CY_U3P_SUCCESS - if the SPI clock is turned off successfully.

See Also

CyU3PSpiSetClock

5.20.4.15 CyU3PReturnStatus_t CyU3PUartSetClock (uint32_t *baudRate*)

Set the frequency for and enable the UART block.

Description

The clock frequency for the UART block depends on the desired output baud rate. This function configures this clock frequency as required and then enables the clock.

The internal clock for the UART block needs to run at 16X the desired output clock frequency. Further, the FX3 device only supports integer and half divisors for deriving the internal clock from the SYS_CLK.

This function sets the internal clock for the UART block to the value that provides the closest approximation of the desired output baud rate.

Return value

CY_U3P_SUCCESS - If the UART clock and baud rate was setup as required.

CY_U3P_ERROR_BAD_ARGUMENT - When invalid arqument is passed to the function.

See Also

CyU3PUartStopClock

Parameters

baudRate	Desired baud rate for the UART interface.

5.20.4.16 CyU3PReturnStatus_t CyU3PUartStopClock (void)

Disable the UART block clock.

Description

This function disables the UART block clock. This should only be called after the UART block has been de-initialized.

Return value

CY_U3P_SUCCESS - if the UART clock was successfully turned off.

See Also

CyU3PUartSetClock

5.21 firmware/u3p_firmware/inc/cyu3mbox.h File Reference

The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PMbox

  Structure that holds a packet of mailbox data.

Typedefs

- typedef struct CyU3PMbox CyU3PMbox

  Structure that holds a packet of mailbox data.
- typedef void(∗ CyU3PMboxCb_t)(CyBool_t mboxEvt)

  Type of function to be called to notify about a mailbox related interrupt.

Functions

- void CyU3PMboxInit (CyU3PMboxCb_t callback)

  Initialize the mailbox handler.
- void CyU3PMboxDeInit (void)

  De-initialize the mailbox handler.
- void CyU3PMboxReset (void)

  Reset the mailbox handler.
- CyU3PReturnStatus_t CyU3PMboxWrite (CyU3PMbox ∗mbox)

  This function sends a mailbox message to the external processor.
- CyU3PReturnStatus_t CyU3PMboxRead (CyU3PMbox ∗mbox)

  This function reads an incoming mailbox message.
- CyU3PReturnStatus_t CyU3PMboxWait (void)

  Wait until the Mailbox register to send messages to P-port is free.

5.21.1 Detailed Description

The mailbox handler is responsible for sending/receiving short messages from an external processor through the mailbox registers. **Description**

The FX3 device implements a set of mailbox registers that can be used to exchange short general purpose messages between FX3 and the external device connected on the P-port. Messages of upto 8 bytes can be sent in each direction at a time. The mailbox handler is responsible for handling mailbox data in both directions.

5.21.2 Typedef Documentation

5.21.2.1 typedef struct CyU3PMbox CyU3PMbox

Structure that holds a packet of mailbox data.

Description

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

5.21.2.2 typedef void(∗ CyU3PMboxCb_t)(CyBool_t mboxEvt)

Type of function to be called to notify about a mailbox related interrupt.

Description

This type is the prototype for a callback function that will be called to notify the application about a mailbox interrupt. The mboxEvt parameter will identify the type of interrupt.

If a read interrupt is received, the mailbox registers have to be read; and if a write interrupt is received, any pending data can be written to the registers.

5.21.3 Function Documentation

5.21.3.1 void CyU3PMboxDeInit (void)

De-initialize the mailbox handler.

Description

Destroys the mailbox related structures.

See Also

CyU3PMboxInit

5.21.3.2 void CyU3PMboxInit (CyU3PMboxCb_t *callback*)

Initialize the mailbox handler.

Description

Initiate the mailbox related structures and register a callback function that will be called on every mailbox related interrupt.

See Also

CyU3PMboxDeInit

Parameters

callback	Callback function to be called on interrupt.

5.21.3.3 CyU3PReturnStatus_t CyU3PMboxRead (CyU3PMbox * *mbox*)

This function reads an incoming mailbox message.

Description

This function is used to read the contents of an incoming mailbox message. This needs to be called in response to a read event callback.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NULL_POINTER - If the mbox pointer provided is NULL.

See Also

CyU3PMboxWrite

Parameters

mbox	Pointer to buffer to hold the incoming message data.

5.21.3.4 void CyU3PMboxReset (void)

Reset the mailbox handler.

Description

Clears the data structures and state related to mailbox handler. Can be used for error recovery.

See Also

CyU3PMboxInit

5.21.3.5 CyU3PReturnStatus_t CyU3PMboxWait (void)

Wait until the Mailbox register to send messages to P-port is free.

Description

This function waits until the mailbox register used to send messages to the processor/device connected on FX3's P-port is free. This is expected to be used in cases where the application needs to ensure that the last message that was sent out has been read by the external processor or device.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_FAILURE - If there is a failure getting a mutex lock on the mailboxes.

See Also

CyU3PMboxWrite

5.21.3.6 CyU3PReturnStatus_t CyU3PMboxWrite (CyU3PMbox * *mbox*)

This function sends a mailbox message to the external processor.

Description

This function writes 8 bytes of data to the outgoing mailbox registers after ensuring that the external processor has read out the previous message. If the previous message has not been read out, this function will block until the registers become free.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NULL_POINTER - If the mbox pointer provided is NULL.

CY_U3P_ERROR_FAILURE - If there is an error getting a mutex lock on the mailboxes.

See Also

CyU3PMboxRead
CyU3PMboxWait

Parameters

mbox	Pointer to mailbox message data.

5.22 firmware/u3p_firmware/inc/cyu3mipicsi.h File Reference

This file contains the MIPI-CSI interface management data structures, functions and macros for CX3 devices.

```
#include <cyu3os.h>
#include <cyu3types.h>
#include <cyu3sib.h>
#include <cyu3gpif.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PMipicsiErrorCounts_t

  Structure defining MIPI-CSI block Phy errors.
- struct CyU3PMipicsiCfg_t

  Structure defining the MIPI-CSI block interface configuration.

Macros

- #define CX3_START_SCK0 0

  The following macros define the various states supported by the fixed function GPIF on the CX3 parts.
- #define **CX3_IDLE_SCK0** 1
- #define **CX3_WAIT_FOR_FRAME_START_SCK0** 2
- #define **CX3_PUSH_DATA_SCK0** 3
- #define **CX3_PUSH_DATA_SCK1** 4
- #define **CX3_WAIT_TO_FILL_SCK0** 5
- #define **CX3_WAIT_TO_FILL_SCK1** 7
- #define **CX3_WAIT_FULL_SCK0_NEXT_SCK1** 6
- #define **CX3_WAIT_FULL_SCK1_NEXT_SCK0** 8
- #define **CX3_PARTIAL_BUFFER_IN_SCK0** 9
- #define **CX3_PARTIAL_BUFFER_IN_SCK1** 10
- #define **CX3_FULL_BUFFER_IN_SCK0** 11
- #define **CX3_FULL_BUFFER_IN_SCK1** 12
- #define **CX3_START_SCK1** 13
- #define **CX3_IDLE_SCK1** 14
- #define **CX3_WAIT_FOR_FRAME_START_SCK1** 15
- #define ALPHA_CX3_START_SCK0 0x0
- #define ALPHA_CX3_START_SCK1 0x0
- #define CX3_NUMBER_OF_STATES 16
- #define CX3_START CX3_START_SCK0
- #define CX3_IDLE CX3_IDLE_SCK0
- #define CX3_WAIT_FOR_FRAME_START CX3_WAIT_FOR_FRAME_START_SCK0
- #define CX3_PUSH_DATA_TO_SCK0 CX3_PUSH_DATA_SCK0
- #define CX3_PUSH_DATA_TO_SCK1 CX3_PUSH_DATA_SCK1
- #define CX3_ALPHA_START ALPHA_CX3_START_SCK0
- #define CY_U3P_CSI_DF_UVC_YUV2 (CY_U3P_CSI_DF_YUV422_8_1)

  Data Format used for the UVC YUV422 (UVC YUV2) format.

Typedefs

- typedef enum CyU3PMipicsiI2cFreq_t CyU3PMipicsiI2cFreq_t

  I2C frequency for MIPI-CSI interface communication.
- typedef enum CyU3PMipicsiSensorIo_t CyU3PMipicsiSensorIo_t

  MIPI Control lines for Image Sensor.

- typedef enum CyU3PMipicsiReset_t CyU3PMipicsiReset_t

  MIPI-CSI interface Reset Type.
- typedef struct CyU3PMipicsiErrorCounts_t CyU3PMipicsiErrorCounts_t

  Structure defining MIPI-CSI block Phy errors.
- typedef enum CyU3PMipicsiDataFormat_t CyU3PMipicsiDataFormat_t

  MIPI-CSI Data Formats.
- typedef enum CyU3PMipicsiPllClkDiv_t CyU3PMipicsiPllClkDiv_t

  Clock Divider Values.
- typedef enum CyU3PMipicsiPllClkFrs_t CyU3PMipicsiPllClkFrs_t

  Frequency range selection for the MIPI-CSI PLL Clock.
- typedef enum CyU3PMipicsiBusWidth_t CyU3PMipicsiBusWidth_t

  Bus Widths supported by the fixed function GPIF interface.
- typedef struct CyU3PMipicsiCfg_t CyU3PMipicsiCfg_t

  Structure defining the MIPI-CSI block interface configuration.

Enumerations

- enum CyU3PMipicsiI2cFreq_t { CY_U3P_MIPICSI_I2C_100KHZ, CY_U3P_MIPICSI_I2C_400KHZ }

  I2C frequency for MIPI-CSI interface communication.
- enum CyU3PMipicsiSensorIo_t { CY_U3P_CSI_IO_XRES = 2, CY_U3P_CSI_IO_XSHUTDOWN = 4 }

  MIPI Control lines for Image Sensor.
- enum CyU3PMipicsiReset_t { CY_U3P_CSI_SOFT_RST = 0, CY_U3P_CSI_HARD_RST }

  MIPI-CSI interface Reset Type.
- enum CyU3PMipicsiDataFormat_t {
  CY_U3P_CSI_DF_RAW8 = 0x00, CY_U3P_CSI_DF_RAW10 = 0x01, CY_U3P_CSI_DF_RAW12 = 0x02, CY_U3P_CSI_DF_RAW14 = 0x08,
  CY_U3P_CSI_DF_RGB888 =0x03, CY_U3P_CSI_DF_RGB666_0 = 0x04, CY_U3P_CSI_DF_RGB666_1 = 0x14, CY_U3P_CSI_DF_RGB565_0 = 0x05,
  CY_U3P_CSI_DF_RGB565_1 = 0x15, CY_U3P_CSI_DF_RGB565_2 = 0x25, CY_U3P_CSI_DF_YUV422_8_0 = 0x06, CY_U3P_CSI_DF_YUV422_8_1 = 0x16,
  CY_U3P_CSI_DF_YUV422_8_2 = 0x26, CY_U3P_CSI_DF_YUV422_10 = 0x07 }

  MIPI-CSI Data Formats.
- enum CyU3PMipicsiPllClkDiv_t { CY_U3P_CSI_PLL_CLK_DIV_8 = 0, CY_U3P_CSI_PLL_CLK_DIV_4, CY_U3P_CSI_PLL_CLK_DIV_2, CY_U3P_CSI_PLL_CLK_DIV_INVALID }

  Clock Divider Values.
- enum CyU3PMipicsiPllClkFrs_t { CY_U3P_CSI_PLL_FRS_500_1000M = 0, CY_U3P_CSI_PLL_FRS_250_500M, CY_U3P_CSI_PLL_FRS_125_250M, CY_U3P_CSI_PLL_FRS_63_125M }

  Frequency range selection for the MIPI-CSI PLL Clock.
- enum CyU3PMipicsiBusWidth_t { CY_U3P_MIPICSI_BUS_8 = 0, CY_U3P_MIPICSI_BUS_16, CY_U3P_MIPICSI_BUS_24 }

  Bus Widths supported by the fixed function GPIF interface.

Functions

- CyU3PReturnStatus_t CyU3PMipicsiInit (void)

  Initialize MIPI-CSI block.
- CyU3PReturnStatus_t CyU3PMipicsiDeInit (void)

  De-Initialize MIPI-CSI block.

- CyU3PReturnStatus_t CyU3PMipicsiSetIntfParams (CyU3PMipicsiCfg_t *csiCfg, CyBool_t wakeOnConfigure)

  Configure MIPI-CSI interface.
- CyU3PReturnStatus_t CyU3PMipicsiQueryIntfParams (CyU3PMipicsiCfg_t *csiCfg)

  Read MIPI-CSI interface configuration from the block.
- CyU3PReturnStatus_t CyU3PMipicsiReset (CyU3PMipicsiReset_t resetType)

  Reset the MIPI-CSI interface.
- void CyU3PCx3DeviceReset (CyBool_t isWarmReset, CyBool_t sensorResetHigh)

  Reset the CX3 Device.
- CyU3PReturnStatus_t CyU3PMipicsiGetErrors (CyBool_t clrErrCnts, CyU3PMipicsiErrorCounts_t *errorCounts)

  Get MIPI-CSI block Phy error counts.
- CyU3PReturnStatus_t CyU3PMipicsiWakeup (void)

  MIPI-CSI block Wakeup.
- CyU3PReturnStatus_t CyU3PMipicsiSleep (void)

  Mipi-CSI block Sleep.
- CyU3PReturnStatus_t CyU3PMipicsiSetSensorControl (CyU3PMipicsiSensorIo_t io, CyBool_t value)

  Sensor XRES and XSHUTDOWN Signals.
- CyBool_t CyU3PMipicsiCheckBlockActive (void)

  Check if MIPI-CSI interface is Active or in Low power sleep.
- CyU3PReturnStatus_t CyU3PMipicsiInitializeGPIO (void)

  Initialize the GPIO block on the CX3.
- CyU3PReturnStatus_t CyU3PMipicsiInitializeI2c (CyU3PMipicsiI2cFreq_t freq)

  Configure and Initialize the I2C Block on the CX3.
- CyU3PReturnStatus_t CyU3PMipicsiInitializePIB (void)

  Initialize and configure the PIB block on the CX3.
- CyU3PReturnStatus_t CyU3PMipicsiGpifLoad (CyU3PMipicsiBusWidth_t busWidth, uint32_t bufferSize)

  Fixed Function GPIF Waveform Load.

5.22.1 Detailed Description

This file contains the MIPI-CSI interface management data structures, functions and macros for CX3 devices.

5.22.2 Macro Definition Documentation

5.22.2.1 #define ALPHA_CX3_START_SCK0 0x0

Initial value of early outputs from the CX3 GPIF state machine.

5.22.2.2 #define ALPHA_CX3_START_SCK1 0x0

Initial value of early outputs from the CX3 GPIF state machine.

5.22.2.3 #define CX3_ALPHA_START ALPHA_CX3_START_SCK0

Mapping for legacy value from SDK 1.3

5.22.2.4 #define CX3_IDLE CX3_IDLE_SCK0

Mapping for legacy value from SDK 1.3

5.22.2.5 #define CX3_NUMBER_OF_STATES 16

Number of states in the CX3 GPIF state machine

5.22.2.6 #define CX3_PUSH_DATA_TO_SCK0 CX3_PUSH_DATA_SCK0

Mapping for legacy value from SDK 1.3

5.22.2.7 #define CX3_PUSH_DATA_TO_SCK1 CX3_PUSH_DATA_SCK1

Mapping for legacy value from SDK 1.3

5.22.2.8 #define CX3_START CX3_START_SCK0

Mapping for legacy value from SDK 1.3

5.22.2.9 #define CX3_START_SCK0 0

The following macros define the various states supported by the fixed function GPIF on the CX3 parts.

5.22.3 Fixed Function GPIF States

5.22.3.1 #define CX3_WAIT_FOR_FRAME_START CX3_WAIT_FOR_FRAME_START_SCK0

Mapping for legacy value from SDK 1.3

5.22.4 Typedef Documentation

5.22.4.1 typedef enum CyU3PMipicsiBusWidth_t CyU3PMipicsiBusWidth_t

Bus Widths supported by the fixed function GPIF interface.

Description

This enumeration defines the bus widths supported by the fixed function GPIF interface on the CX3 parts. CX3 supports bus widths of 8-Bits, 16-Bits and 24-Bits.

Note

The DMA buffer size being used to transfer data from the GPIF interface must be aligned to the data width of the interface used. The data buffer size in bytes must be a multiple of 16 for 16-Bit interfaces and a multiple of 24 for 24-Bit buffers.

See Also

CyU3PMipicsiGpifLoad

5.22.4.2 typedef struct CyU3PMipicsiCfg_t CyU3PMipicsiCfg_t

Structure defining the MIPI-CSI block interface configuration.

Description

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

Note

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

See Also

CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams
CyU3PMipicsiPllClkFrs_t
CyU3PMipicsiPllClkDiv_t

5.22.4.3 typedef enum CyU3PMipicsiDataFormat_t CyU3PMipicsiDataFormat_t

MIPI-CSI Data Formats.

Description

This enumerated type lists the MIPI CSI data formats supported by the MIPI-CSI block on the CX3. The MIPI-CSI block on the CX3 supports the listed data formats in 8, 16 and 24 bit modes. Some data formats (RAW8, RGB888) support only a fixed data width, while some formats support more than one data widths with different padding mechanisms and byte orders on the received data.

Note

The user needs to set the Fixed Function GPIF interface on the CX3 to an appropriate width based on the Data Format being selected. Selection of an incorrect GPIF bus width can lead to loss of data or introduction of garbage data into the data stream.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

5.22.4.4 typedef struct CyU3PMipicsiErrorCounts_t CyU3PMipicsiErrorCounts_t

Structure defining MIPI-CSI block Phy errors.

Description

The following structure is used to fetch count of MIPI-CSI Phy level errors from the MIPI-CSI block on the CX3.

See Also

CyU3PMipicsiGetErrors

5.22.4.5 typedef enum CyU3PMipicsiI2cFreq_t CyU3PMipicsiI2cFreq_t

I2C frequency for MIPI-CSI interface communication.

Description

This enumeration defines the I2C frequency for communication with the Image Sensor and the MIPI-CSI interface. The CX3 part supports communication at 100KHz and 400 KHz.

See Also

CyU3PMipicsiInitializeI2c

5.22.4.6 typedef enum CyU3PMipicsiPllClkDiv_t CyU3PMipicsiPllClkDiv_t

Clock Divider Values.

Description

This enumerated type lists Clock divider values permitted for the various clocks on the MIPI-CSI block of the CX3 device. The clocks on MIPI-CSI block are derived from a primary PLL clock which is generated using 19.2 MHz System Reference Clock.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

5.22.4.7 typedef enum CyU3PMipicsiPllClkFrs_t CyU3PMipicsiPllClkFrs_t

Frequency range selection for the MIPI-CSI PLL Clock.

Description

This enumeration is used to define the frequency rage in which the PLL clock on the MIPI-CSI block is operating.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

5.22.4.8 typedef enum CyU3PMipicsiReset_t CyU3PMipicsiReset_t

MIPI-CSI interface Reset Type.

Description

This enumerated type lists the reset types supported for the MIPI-CSI interface block on the CX3. The MIPI-CSI interface provides two reset modes - Hard reset, which power cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block.

See Also

CyU3PMipicsiReset

5.22.4.9 typedef enum CyU3PMipicsiSensorIo_t CyU3PMipicsiSensorIo_t

MIPI Control lines for Image Sensor.

Description

This enumeration defines the IO lines for MIPI XReset and XShutdown signals. The user drives these lines high or low using the CyU3PMipicsiSetSensorControl function

See Also

CyU3PMipicsiSetSensorControl

5.22.5 Enumeration Type Documentation

5.22.5.1 enum CyU3PMipicsiBusWidth_t

Bus Widths supported by the fixed function GPIF interface.

Description

This enumeration defines the bus widths supported by the fixed function GPIF interface on the CX3 parts. CX3 supports bus widths of 8-Bits, 16-Bits and 24-Bits.

Note

The DMA buffer size being used to transfer data from the GPIF interface must be aligned to the data width of the interface used. The data buffer size in bytes must be a multiple of 16 for 16-Bit interfaces and a multiple of 24 for 24-Bit buffers.

See Also

CyU3PMipicsiGpifLoad

Enumerator

CY_U3P_MIPICSI_BUS_8 Use an 8-Bit data bus

CY_U3P_MIPICSI_BUS_16 Use a 16-Bit data bus

CY_U3P_MIPICSI_BUS_24 Use a 24-Bit data bus

5.22.5.2 enum CyU3PMipicsiDataFormat_t

MIPI-CSI Data Formats.

Description

This enumerated type lists the MIPI CSI data formats supported by the MIPI-CSI block on the CX3. The MIPI-CSI block on the CX3 supports the listed data formats in 8, 16 and 24 bit modes. Some data formats (RAW8, RGB888) support only a fixed data width, while some formats support more than one data widths with different padding mechanisms and byte orders on the received data.

Note

The user needs to set the Fixed Function GPIF interface on the CX3 to an appropriate width based on the Data Format being selected. Selection of an incorrect GPIF bus width can lead to loss of data or introduction of garbage data into the data stream.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

Enumerator

CY_U3P_CSI_DF_RAW8 RAW8 Data Type. CSI-2 Data Type 0x2A
8 Bit Output = RAW[7:0]

CY_U3P_CSI_DF_RAW10 RAW10 Data Type. CSI-2 Data Type 0x2B
16 Bit Output = 6'b0,RAW[9:0]

CY_U3P_CSI_DF_RAW12 RAW12 Data Type. CSI-2 Data Type 0x2C
16 Bit Output = 4'b0,RAW[11:0]

CY_U3P_CSI_DF_RAW14 RAW14 Data Type. CSI-2 Data Type 0x2D
16 Bit Output = 2'b0,RAW[13:0]

CY_U3P_CSI_DF_RGB888 RGB888 Data type. CSI-2 Data Type 0x24.
24 Bit Output = R[7:0],G[7:0],B[7:0]

CY_U3P_CSI_DF_RGB666_0 RGB666 Data type. CSI-2 Data Type 0x23.
24 Bit Output = 2'b0,R[5:0],2'b0,G[5:0], 2'b0,B[5:0]

CY_U3P_CSI_DF_RGB666_1 RGB666 Data type. CSI-2 Data Type 0x23.
24 Bit Output = 6'b0,R[5:0],G[5:0], B[5:0]

CY_U3P_CSI_DF_RGB565_0 RGB565 Data type. CSI-2 Data Type 0x22.
24 Bit Output = 2'b0,R[4:0],3'b0,G[5:0], 2'b0,B[4:0],1'b0

CY_U3P_CSI_DF_RGB565_1 RGB565 Data type. CSI-2 Data Type 0x22.
24 Bit Output = 3'b0,R[4:0],2'b0,G[5:0], 3'b0,B[4:0]

CY_U3P_CSI_DF_RGB565_2 RGB565 Data type. CSI-2 Data Type 0x22.
24 Bit Output = 8'b0,R[4:0],G[5:0], B[4:0]
16 Bit Output = R[4:0],G[5:0], B[4:0]

CY_U3P_CSI_DF_YUV422_8_0 YUV422 8-Bit Data type. CSI-2 Type 0x1E.
24 Bit Output = 16'b0,P[7:0]
16 Bit Output = 8'b0, P[7:0]
8 Bit Output = P[7:0]
Data Order: U1,Y1,V1,Y2,U3,Y3,V3,Y4....

CY_U3P_CSI_DF_YUV422_8_1 YUV422 8-Bit Data type. CSI-2 Type 0x1E.
24 Bit Output = 8'b0,P[15:0]
16 Bit Output = P[15:0]
Data Order: {U1,Y1},{V1,Y2},{U3,Y3},{V3,Y4}....

CY_U3P_CSI_DF_YUV422_8_2 YUV422 8-Bit Data type. CSI-2 Type 0x1E.
24 Bit Output = 8'b0,P[15:0]
16 Bit Output = P[15:0]
Data Order: {Y1,U1},{Y2,V1},{Y3,U3},{Y4,V3}....

CY_U3P_CSI_DF_YUV422_10 YUV422 10-Bit Data type. CSI-2 Type 0x1F.
24 Bit Output = 14'b0,P[9:0]
16 Bit Output = 6'b0,P[9:0]
Data Order: U1,Y1,V1,Y2,U3,Y3,V3,Y4....

5.22.5.3 enum CyU3PMipicsiI2cFreq_t

I2C frequency for MIPI-CSI interface communication.

Description

This enumeration defines the I2C frequency for communication with the Image Sensor and the MIPI-CSI interface. The CX3 part supports communication at 100KHz and 400 KHz.

See Also

CyU3PMipicsiInitializeI2c

Enumerator

CY_U3P_MIPICSI_I2C_100KHZ 100 KHz operation

CY_U3P_MIPICSI_I2C_400KHZ 400 KHz operation

5.22.5.4 enum CyU3PMipicsiPllClkDiv_t

Clock Divider Values.

Description

This enumerated type lists Clock divider values permitted for the various clocks on the MIPI-CSI block of the CX3 device. The clocks on MIPI-CSI block are derived from a primary PLL clock which is generated using 19.2 MHz System Reference Clock.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

Enumerator

CY_U3P_CSI_PLL_CLK_DIV_8 Clk = PLL_ClK/8
CY_U3P_CSI_PLL_CLK_DIV_4 Clk = PLL_ClK/8
CY_U3P_CSI_PLL_CLK_DIV_2 Clk = PLL_ClK/8
CY_U3P_CSI_PLL_CLK_DIV_INVALID Invalid Value

5.22.5.5 enum CyU3PMipicsiPllClkFrs_t

Frequency range selection for the MIPI-CSI PLL Clock.

Description

This enumeration is used to define the frequency rage in which the PLL clock on the MIPI-CSI block is operating.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiSetIntfParams
CyU3PMipicsiQueryIntfParams

Enumerator

CY_U3P_CSI_PLL_FRS_500_1000M PLL Clock output range 500MHz-1GHz
CY_U3P_CSI_PLL_FRS_250_500M PLL Clock output range 250-500MHz
CY_U3P_CSI_PLL_FRS_125_250M PLL Clock output range 125-250MHz
CY_U3P_CSI_PLL_FRS_63_125M PLL Clock output range 62.5-125MHz

5.22.5.6 enum CyU3PMipicsiReset_t

MIPI-CSI interface Reset Type.

Description

This enumerated type lists the reset types supported for the MIPI-CSI interface block on the CX3. The MIPI-CSI interface provides two reset modes - Hard reset, which power cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block.

See Also

CyU3PMipicsiReset

Enumerator

CY_U3P_CSI_SOFT_RST Perform a soft reset on the MIPI-CSI interface block
CY_U3P_CSI_HARD_RST Perform a hard reset on the MIPI-CSI interface block

5.22.5.7 enum CyU3PMipicsiSensorIo_t

MIPI Control lines for Image Sensor.

Description

This enumeration defines the IO lines for MIPI XReset and XShutdown signals. The user drives these lines high or low using the CyU3PMipicsiSetSensorControl function

See Also

CyU3PMipicsiSetSensorControl

Enumerator

CY_U3P_CSI_IO_XRES Image Sensor Reset IO (XRES)

CY_U3P_CSI_IO_XSHUTDOWN Image Sensor Shutdown IO (XSHUTDOWN)

5.22.6 Function Documentation

5.22.6.1 void CyU3PCx3DeviceReset (CyBool_t *isWarmReset,* CyBool_t *sensorResetHigh*)

Reset the CX3 Device.

Description

This function is similar to the CyU3PDeviceReset function used for FX3/FX3S devices. In addition to resetting the CX3 device, this function also drives the XRES line output to reset the Image Sensor before the CX3 reset.

Return value

None as this function does not return.

See Also

CyU3PDeviceReset

Parameters

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 CX3 needs to be performed again.
sensorReset-High	True if Sensor is to be reset by driving XRES High, False if Sensor reset needs XRES as low.

5.22.6.2 CyBool_t CyU3PMipicsiCheckBlockActive (void)

Check if MIPI-CSI interface is Active or in Low power sleep.

Description

This function is used to check if the MIPI-CSI interface is Active or in low power sleep.

Return value

CyTrue - If the interface block is Active.

CyFalse - If the interface block is in Low power sleep

5.22.6.3 CyU3PReturnStatus_t CyU3PMipicsiDeInit (void)

De-Initialize MIPI-CSI block.

Description

This function de-initializes MIPI-CSI interface block on the CX3 device and is expected to be called prior to calling CyU3PSysEnterStandbyMode. This function should be called before the I2C block or GPIO blocks are de-initialized. MIPI XReset and XShutdown signals shall not be driven by the CX3 after this call.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

See Also

CyU3PSysEnterStandbyMode

5.22.6.4 CyU3PReturnStatus_t CyU3PMipicsiGetErrors (CyBool_t *clrErrCnts*, CyU3PMipicsiErrorCounts_t * *errorCounts*)

Get MIPI-CSI block Phy error counts.

Description

This function is used to get a count of CSI protocol and physical layer errors from the MIPI-CSI block. The function takes a parameter which determines whether or not the error counts on the interface are cleared. The error counts for each error type is retrieved via an pointer of type CyU3PMipicsiErrorCounts_t passed to this function. The error count values for each type can reach a maximum count of 0xFF. The count values will continue to report the existing error value on each call unless the function explicitly clears the counts using clrErrCnts.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized.

CY_U3P_ERROR_NULL_POINTER - If the clrErrCnts object is uninitialized.

See Also

CyU3PMipicsiErrorCounts_t

Parameters

clrErrCnts	Set to CyTrue to clear the error counts
errorCounts	Error Counts

5.22.6.5 CyU3PReturnStatus_t CyU3PMipicsiGpifLoad (CyU3PMipicsiBusWidth_t *busWidth*, uint32_t *bufferSize*)

Fixed Function GPIF Waveform Load.

Description

The CX3 has a fixed function GPIF interface designed for Image sensor data acquisition from the MIPI-CSI interface. This function allows selection of the GPIF data bus-width and configuration of the size of the DMA buffer provided for GPIF transfers. The PIB block should have been initialized prior to calling this function.

Return value

CY_U3P_SUCCESS - If the operation is successful CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_BAD_ARGUMENT - If an invalid argument is passed to the function.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_ALREADY_STARTED - If the GPIF hardware is running.

See Also

CyU3PMipicsiBusWidth_t
CyU3PMipicsiInitializePIB

Parameters

busWidth	Load GPIF State machine and settings for selected busWidth
bufferSize	Set Data counter based on DMA buffer size being used. This value MUST match the size of the buffer actual data buffer being used i.e. (The size being passed to the CyU3PDmaChannel-Create API) - (The Size of Header and Footer) e.g. If dmaCfg is CyU3PDmaChannelConfig_t object being passed to the ChannelCreate API the value of bufferSize = dmaCfg.size - (dma-Cfg.prodHeader + dmaCfg.prodFooter)

5.22.6.6 CyU3PReturnStatus_t CyU3PMipicsiInit (void)

Initialize MIPI-CSI block.

Description

This function initializes MIPI-CSI interface block on the CX3 device and is expected to be called prior to any other calls to the MIPI-CSI block. The GPIO Block, the PIB block and the I2C blocks should have been initialized prior to calling this function. Helper functions CyU3PMipicsiInitializeGPIO(), CyU3PMipicsiInitializePIB() and CyU3P-MipicsiInitializeI2c() have been provided to initialize these blocks to the default values to be used for the MIPI-CSI block. Please see documentation on each of those functions for more information on the specific settings.

The CX3 part uses the I2C block to communicate with both the MIPI-CSI block and any connected Image Sensor. The 7 bit I2C slave address 8b'0000111X (Read 0x0F, Write 0x0E) is used internally by the CX3 part to configure and control the MIPI-CSI interface block and should not be used by any external I2C slaves connected to the CX3 on the I2C bus.

Note

This function sets the default state of the MIPI XReset and XShutdown signals as Drive 0. If either of these signals, needs to be in Drive 1 state for sensor operation, it needs to be explictily set to Drive 1 state using CyU3PMipicsi-SetSensorControl() after calling CyU3PMipicsiInit(). This call leaves the interface in a low-power mode and CyU3-PMipicsiWakeup() should be called to enable the clocks on the MIPI-CSI interface.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

See Also

CyU3PMipicsiInitializeI2c
CyU3PMipicsiInitializeGPIO
CyU3PMipicsiInitializePIB
CyU3PCx3DeviceReset

CyU3PMipicsiSetSensorControl

5.22.6.7 CyU3PReturnStatus_t CyU3PMipicsiInitializeGPIO (void)

Initialize the GPIO block on the CX3.

Description

This function is a helper routine to initialize the GPIO block with default values for the CX3 device.

The function internally calls CyU3PGpioInit with the following parameters:

CyU3PGpioClock_t settings:

fastClkDiv = 2;

slowClkDiv = 0;

simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2;

clkSrc = CY_U3P_SYS_CLK;

halfDiv = 0;

Callback set to NULL.

This function should be called before initializing the PIB block or calling CyU3PMipicsiInit(). In case different parameters are required, the application can use its own code to initialize the GPIO block prior to initializing the PIB block or calling CyU3PMipicsiInit().

Return value

CY_U3P_SUCCESS - If the GPIO initialization is successful CY_U3P_ERROR_ALREADY_STARTED - If the GPIO block has already been initialized

See Also

CyU3PGpioInit

5.22.6.8 CyU3PReturnStatus_t CyU3PMipicsiInitializeI2c (CyU3PMipicsiI2cFreq_t *freq*)

Configure and Initialize the I2C Block on the CX3.

Description

This function is a helper routine to initialize the I2C block to either 100KHz or 400KHz for the CX3 device.

The function internally calls CyU3PI2cInit and then calls CyU3PI2cSetConfig to configure the I2C block with the following parameters:

CyU3PI2cConfig_t settings:

bitRate = 100000 or 400000 depending on freq

isDma = CyFalse;

busTimeout = 0xFFFFFFFF

dmaTimeout = 0xFFFF;

Callback is set to NULL.

The I2C block needs to be initialized prior to calling CyU3PMipicsiInit(). In case different parameters are required, the application can use its own code to initialize the I2C block prior to calling CyU3PMipicsiInit() but must ensure that the block is initialized to either 100KHz or 400KHz.

Note

If the I2C block is already initialized before this function is called, it will override the current configuration of the I2C block with the configuration specified in the description of this function.

Return value

CY_U3P_SUCCESS - If the initialization is successful CY_U3P_ERROR_NOT_CONFIGURED - When I2C has not been enabled in IO configuration. CY_U3P_ERROR_TIMEOUT - When there is timeout happening during configuration CY_U3P_ERROR_MUTEX_FAILURE- When there is a failure in acquiring a mutex lock

See Also

CyU3PI2cInit
CyU3PI2cSetConfig

Parameters

freq	Frequency 100KHz or 400KHz

5.22.6.9 CyU3PReturnStatus_t CyU3PMipicsiInitializePIB (void)

Initialize and configure the PIB block on the CX3.

Description

This function is a helper routine to initialize the PIB block on the CX3 part used by the fixed function GPIF .

The function internally calls CyU3PPibInit() with the following parameters:

CyU3PPibClock_t settings:

clkDiv = 2;

clkSrc = CY_U3P_SYS_CLK

isHalfDiv = CyFalse;

isDllEnable = CyFalse;

doInit is set to CyTrue.

The PIB block needs to be initialized, prior to calling CyU3PMipicsiGpifLoad(), to configure the fixed function GPIF for the CX3 part.

Return value

CY_U3P_SUCCESS - If the operation is successful

See Also

CyU3PPibInit

5.22.6.10 CyU3PReturnStatus_t CyU3PMipicsiQueryIntfParams (CyU3PMipicsiCfg_t * *csiCfg*)

Read MIPI-CSI interface configuration from the block.

Description

This function is used to read back the MIPI-CSI interface parameters from the block. The parameters read back are provided to the calling function via the pointer of type CyU3PMipicsiCfg_t passed in from the calling function.

Note

Memory for the pointer csiCfg must be allocated in the calling function.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_BAD_ARGUMENT - If an invalid argument is read from the interface.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized.

CY_U3P_ERROR_NULL_POINTER - If the csiCfg object is uninitialized.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiQueryIntfParams

Parameters

csiCfg	Configuration Data read back from the MIPI-CSI interface

5.22.6.11 CyU3PReturnStatus_t CyU3PMipicsiReset (CyU3PMipicsiReset_t *resetType*)

Reset the MIPI-CSI interface.

Description

This function is used to reset the MIPI-CSI block on the CX3. The MIPI-CSI block provides two reset modes - Hard reset, which power-cycles the entire block, and Soft reset, which does not reset the I2C communication channel used by the block. The reset mode is selcted via the parameter of type CyU3PMipicsiReset_t passed to this function.

Note

The Soft Reset cannot be called on the interface until the block has been initialized. A Hard reset however, can be called on the interface at any point in time. The CyU3PMipicsiInit() call internally performs a Hard reset on the interface before initializing it. A Hard reset sets the default state of the MIPI XReset and XShutdown signals as Drive 0. If either of these signals, needs to be in Drive 1 state for sensor operation, it needs to be explictily set to Drive 1 state using CyU3PMipicsiSetSensorControl() after calling CyU3PMipicsiReset(CY_U3P_CSI_HARD_RST). A Soft reset does not change the state of the XReset and XShutdown signals.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_BAD_ARGUMENT - If an invalid argument is passed to the function.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized (Soft Reset Only).

See Also

CyU3PMipicsiInit
CyU3PMipicsiReset_t

Parameters

resetType	Reset Type

5.22.6.12 CyU3PReturnStatus_t CyU3PMipicsiSetIntfParams (CyU3PMipicsiCfg_t * *csiCfg*, CyBool_t *wakeOnConfigure*)

Configure MIPI-CSI interface.

Description

This function is used to configure the MIPI-CSI interface parameters over I2C. The function takes in an object of the type CyU3PMipicsiCfg_t and configures the MIPI-CSI interface. The function powers off the interface clocks before changing/setting the configuration. Parameter wakeOnConfigure is used to turn the clocks ON immediately after the configuration has been completed or to leave the clocks powered down. If the clocks are left powered down, CyU3PMipicsiWakeup() should be called to start the clocks. The state of the interface (Active or Powered down) can be determined using the CyU3PMipicsiCheckBlockActive() call.

Note

This function can be called when the interface has been put into low power mode using the CyU3PMipicsiSleep() function.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_BAD_ARGUMENT - If an invalid argument is passed to this function.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized.

CY_U3P_ERROR_NULL_POINTER - If the csiCfg object is uninitialized.

See Also

CyU3PMipicsiCfg_t
CyU3PMipicsiQueryIntfParams
CyU3PMipicsiCheckBlockActive
CyU3PMipicsiWakeup

Parameters

csiCfg	Configuration Data to be set to the MIPI-CSI interface
wakeOnConfigure	Start the MIPI-CSI interface clocks immediately after configuring If wakeOnConfigure is CyFalse, CyU3PMipicsiWakeup() must be called to start the clocks when ready for transfers.

5.22.6.13 CyU3PReturnStatus_t CyU3PMipicsiSetSensorControl (CyU3PMipicsiSensorIo_t *io*, CyBool_t *value*)

Sensor XRES and XSHUTDOWN Signals.

Description

This function is used to drive the XRES and XSHUTDOWN signals from the CX3 to Image sensor. The function allows for the signals to be driven high or low. The function can drive either one of the two signals or both signals (both driven high or both driven low) simultaneously. To set both signals to the same value use the mask (CY_U3P_CSI_IO_XRES | CY_U3P_CSI_IO_XSHUTDOWN) as value for io. To set the two signals to separate values multiple calls to this function are required (once for each signal).

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_BAD_ARGUMENT - If an invalid argument is passed to the function.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized (Soft Reset Only).

See Also

CyU3PMipicsiSensorIo_t

Parameters

io	IO to be driven.
value	CyTrue to drive Signal High, CyFalse for Low.

5.22.6.14 CyU3PReturnStatus_t CyU3PMipicsiSleep (void)

Mipi-CSI block Sleep.

Description

This function is used to disable the PLL clocks on the MIPI-CSI interface block and place it in low-power sleep. No data transfers from the Image Sensor to the CX3 will occur while the block is in low power sleep mode.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized.

See Also

CyU3PMipicsiWakeup

5.22.6.15 CyU3PReturnStatus_t CyU3PMipicsiWakeup (void)

MIPI-CSI block Wakeup.

Description

This function is used enable the clocks on the MIPI-CSI interface block to take it from Low power sleep to Active.

Return value

CY_U3P_SUCCESS - If the operation completes successfully.

CY_U3P_ERROR_TIMEOUT - If an I2C read/write timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED - If the part on which this function is called is not a CX3 device.

CY_U3P_ERROR_UNKNOWN - If a general or unknown error occurred during the operation.

CY_U3P_ERROR_NOT_STARTED - If the interface has not been initialized.

See Also

CyU3PMipicsiSleep

5.23 firmware/u3p_firmware/inc/cyu3mmu.h File Reference

Cache and Memory Management for the FX3 firmware.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Macros

- #define CYU3P_ITCM_BASE_ADDR (0x00000000)
- #define CYU3P_ITCM_SIZE (0x00004000)
- #define CYU3P_ITCM_SZ_EN (0x00000015)
- #define CYU3P_DTCM_BASE_ADDR (0x10000000)
- #define CYU3P_DTCM_SIZE (0x00002000)
- #define CYU3P_DTCM_SZ_EN (0x00000011)
- #define CYU3P_SYSMEM_BASE_ADDR (0x40000000)
- #define CYU3P_SYSMEM_SIZE (0x00080000)
- #define CYU3P_MMIO_BASE_ADDR (0xE0000000)
- #define CYU3P_MMIO_SIZE (0x10000000)
- #define CYU3P_ROM_BASE_ADDR (0xF0000000)
- #define CYU3P_ROM_SIZE (0x00008000)
- #define CYU3P_VIC_BASE_ADDR (0xFFFFF000)
- #define CYU3P_VIC_SIZE (0x00001000)
- #define CYU3P_GCTL_PAGE_TABLE_ADDR (0xE0058000)
- #define CYU3P_CACHE_LINE_SZ (5)
- #define CYU3P_CACHE_SIZE (13)
- #define CYU3P_CACHE_NWAYS (2)
- #define CYU3P_CACHE_WAY_SZ (CYU3P_CACHE_SIZE - CYU3P_CACHE_NWAYS - CYU3P_CACHE_LINE_SZ)

  Log(2) of the size of a cache way in bytes.
- #define CYU3P_ICACHE_EN_MASK (0x1000)
- #define CYU3P_DCACHE_EN_MASK (0x04)
- #define CYU3P_MMU_EN_MASK (0x01)
- #define CYU3P_CACHE_MMU_EN_MASK (0x1005)
- #define CYU3P_CACHE_REPLACEMENT_MASK (0x4000)
- #define CYU3P_TCMREG_ADDRESS_MASK (0xFFFFF000)

Functions

- void CyU3PSysDisableICache (void)

  Disable the instruction cache.
- void CyU3PSysDisableDCache (void)

  Disable the data cache.
- void CyU3PSysDisableMMU (void)

  Disable the MMU.
- void CyU3PSysDisableCacheMMU (void)

  Disable the caches and MMU.
- void CyU3PSysEnableICache (void)

  Enable the instruction cache.
- void CyU3PSysEnableDCache (void)

  Enable the Data Cache.

- void CyU3PSysEnableMMU (void)

  Enable the MMU.
- void CyU3PSysEnableCacheMMU (void)

  Enable the Caches and the MMU.
- void CyU3PSysFlushCaches (void)

  Flush both I and D Caches.
- void CyU3PSysFlushICache (void)

  Flush the I-Cache.
- void CyU3PSysFlushDCache (void)

  Flush the D-Cache.
- void CyU3PSysCleanDCache (void)

  Clean the entire D-Cache.
- void CyU3PSysClearDCache (void)

  Clean and Flush the entire D-Cache.
- void CyU3PSysFlushIRegion (uint32_t *addr, uint32_t len)

  Flush a code region from the I-Cache.
- void CyU3PSysFlushDRegion (uint32_t *addr, uint32_t len)

  Flush a data region from the D-Cache.
- void CyU3PSysCleanDRegion (uint32_t *addr, uint32_t len)

  Clean a data region from the D-Cache.
- void CyU3PSysClearDRegion (uint32_t *addr, uint32_t len)

  Clean and flush a data region from the D-Cache.
- void CyU3PSysBarrierSync (void)

  Memory barrier synchronization function.
- uint32_t CyU3PSysCacheIRegion (uint32_t *addr, uint32_t len)

  Function to lock a code region into the I-Cache.
- uint32_t CyU3PSysCacheDRegion (uint32_t *addr, uint32_t len)

  Function to lock a data region into the D-Cache.
- void CyU3PSysInitTCMs (void)

  Function to initialize the TCM regions.
- void CyU3PInitPageTable (void)

  Function to create the page tables for the device memory map.
- void CyU3PSysLockTLBEntry (uint32_t *addr)

  Function to lock the TLB entry for a page walk.
- void CyU3PSysFlushTLBEntry (uint32_t *addr)

  Invalidate the TLB entry corresponding to a specified MVA.
- void CyU3PSysLoadTLB (void)

  Function to lock all valid page walks into TLB.

5.23.1 Detailed Description

Cache and Memory Management for the FX3 firmware. **Description**

The FX3 device has 8KB of I-cache and 8KB of D-cache. The default cache handling is done internal to the library. The cache and MMU are initialized. The cache control should be done using CyU3PDeviceCacheControl API. This section explains the extended cache functions that can be used for customized / advanced cache handling. Most of these functions are standard implementations for the ARM926EJ-S core.

If the data cache is enabled, then all buffers used for DMA operation has to be 32 byte aligned and 32 byte multiple. This is because the size of the cache line in the core is 32 bytes. The DMA buffers include all buffers used with DMA APIs as well as used with library APIs like CyU3PUsbSetDesc, CyU3PUsbSendEP0Data, CyU3PUsbGetEP0Data, CyU3PUsbHostSendSetupRqt etc.

5.23.2 Macro Definition Documentation

5.23.2.1 #define CYU3P_CACHE_LINE_SZ (5)

Log(2) of cache line size in bytes.

5.23.2.2 #define CYU3P_CACHE_MMU_EN_MASK (0x1005)

Mask for Cache and MMU enable bits.

5.23.2.3 #define CYU3P_CACHE_NWAYS (2)

Log(2) of number of cache ways.

5.23.2.4 #define CYU3P_CACHE_REPLACEMENT_MASK (0x4000)

Mask for Cache replacement policy bit in cp15:c1

5.23.2.5 #define CYU3P_CACHE_SIZE (13)

Log(2) of cache size in bytes.

5.23.2.6 #define CYU3P_DCACHE_EN_MASK (0x04)

Mask for D-Cache enable bit in cp15:c1

5.23.2.7 #define CYU3P_DTCM_BASE_ADDR (0x10000000)

D-TCM base address.

5.23.2.8 #define CYU3P_DTCM_SIZE (0x00002000)

D-TCM size in bytes.

5.23.2.9 #define CYU3P_DTCM_SZ_EN (0x00000011)

D-TCM size in the format for cp15:c9

5.23.2.10 #define CYU3P_GCTL_PAGE_TABLE_ADDR (0xE0058000)

Address of MMIO mapped page table.

5.23.2.11 #define CYU3P_ICACHE_EN_MASK (0x1000)

Mask for I-Cache enable bit in cp15:c1

5.23.2.12 #define CYU3P_ITCM_BASE_ADDR (0x00000000)

I-TCM base address.

5.23.2.13 #define CYU3P_ITCM_SIZE (0x00004000)

I-TCM size in bytes.

5.23.2.14 #define CYU3P_ITCM_SZ_EN (0x00000015)

I-TCM size in the format for cp15:c9

5.23.2.15 #define CYU3P_MMIO_BASE_ADDR (0xE0000000)

MMIO base address.

5.23.2.16 #define CYU3P_MMIO_SIZE (0x10000000)

MMIO size in bytes.

5.23.2.17 #define CYU3P_MMU_EN_MASK (0x01)

Mask for MMU enable bit in cp15:c1

5.23.2.18 #define CYU3P_ROM_BASE_ADDR (0xF0000000)

BootROM base address.

5.23.2.19 #define CYU3P_ROM_SIZE (0x00008000)

BootROM size in bytes.

5.23.2.20 #define CYU3P_SYSMEM_BASE_ADDR (0x40000000)

SYSMEM base address.

5.23.2.21 #define CYU3P_SYSMEM_SIZE (0x00080000)

SYSMEM size in bytes.

5.23.2.22 #define CYU3P_TCMREG_ADDRESS_MASK (0xFFFFF000)

Mask for TCM base address in cp15:c9

5.23.2.23 #define CYU3P_VIC_BASE_ADDR (0xFFFFF000)

VIC base address.

5.23.2.24 #define CYU3P_VIC_SIZE (0x00001000)

VIC size in bytes.

5.23.3 Function Documentation

5.23.3.1 void CyU3PInitPageTable (void)

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

Description

This function uses the GCTL registers to create a one-level (section entries only) page table for the FX3 device memory map. The function is not expected to be explicitly invoked and is called by the library during initialization. It is provided for debug purposes.

Return value

None

5.23.3.2 void CyU3PSysBarrierSync (void)

Memory barrier synchronization function.

Description

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

Return value

None

5.23.3.3 uint32_t CyU3PSysCacheDRegion (uint32_t * *addr*, uint32_t *len*)

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

Description

Function to load a specified data region into the D-cache and lock it. A maximum of 3 lock operations can be called, with the size of the data region limited to under 2 KB in each case. The caller should ensure that the D-Cache has been flushed, to avoid the possibility of the content already being in the cache.

Return value

The way index into which the region was locked.

Parameters

addr	Start address of the region to cache.
len	Length of the region in bytes.

5.23.3.4 uint32_t CyU3PSysCacheIRegion (uint32_t * *addr*, uint32_t *len*)

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

Description

Function to load a specified code region into the I-cache and lock it. A maximum of 3 lock operations can be called, with the size of the code region limited to under 2 KB in each case. The caller should ensure that the I-Cache has been flushed, to avoid the possibility of the content already being in the cache.

Return value

The way index into which the region was locked.

Parameters

addr	Start address of the region to cache.
len	Length of the region in bytes.

5.23.3.5 void CyU3PSysCleanDCache (void)

Clean the entire D-Cache.

Description

Function to clean the entire D-Cache. The Test/Clean functionality of the Arm 926 core is used here. The Cache is not flushed by this function.

Return value

None

5.23.3.6 void CyU3PSysCleanDRegion (uint32_t * *addr,* uint32_t *len*)

Clean a data region from the D-Cache.

Description

Function to clean a specified data region from the Data Cache. If the D-cache is enabled and the cache is handled by the application, then this API should be invoked before sending any data using DMA.

Return value

None

Parameters

addr	Start address of the region to clean.
len	Length of the region in bytes.

5.23.3.7 void CyU3PSysClearDCache (void)

Clean and Flush the entire D-Cache.

Description

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

Return value

None

5.23.3.8 void CyU3PSysClearDRegion (uint32_t * *addr,* uint32_t *len*)

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

Description

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

Return value

None

Parameters

addr	Start address of the region to flush.
len	Length of the region in bytes.

5.23.3.9 void CyU3PSysDisableCacheMMU (void)

Disable the caches and MMU.

Description

Function to disable the Instruction and Data caches as well as the MMU. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes only.

Return value

None

See Also

CyU3PSysDisableCacheMMU
CyU3PSysDisableDCache
CyU3PSysDisableICache

5.23.3.10 void CyU3PSysDisableDCache (void)

Disable the data cache.

Description

Function to disable the Data Cache. Should be called from a privileged mode, after ensuring that the D-Cache has been cleaned and flushed. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The D-cache is kept disabled by default at firmware start-up.

Return value

None

See Also

CyU3PSysDisableCacheMMU
CyU3PSysEnableDCache

5.23.3.11 void CyU3PSysDisableICache (void)

Disable the instruction cache.

Description

Function to disable the Instruction Cache. Should be called from a privileged mode, after ensuring that the I-Cache has been flushed. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The I-cache is kept disabled by default at firmware start-up.

Return value

None

See Also

CyU3PSysDisableCacheMMU
CyU3PSysEnableICache

5.23.3.12 void CyU3PSysDisableMMU (void)

Disable the MMU.

Description

Function to disable the Memory Management Unit. All memory will be accessed through physical addresses once MMU has been disabled. Caller should ensure that caches have been flushed and disabled before disabling the MMU. This function should not be explicitly called and is invoked from the library. This function is provided for debug purposes only.

Return value

None

See Also

CyU3PSysDisableCacheMMU
CyU3PSysEnableMMU

5.23.3.13 void CyU3PSysEnableCacheMMU (void)

Enable the Caches and the MMU.

Description

Function to enable the Instruction and Data caches as well as the MMU. Sets up both caches to use random replacement strategy. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes.

Return value

None

See Also

CyU3PSysDisableCacheMMU
CyU3PSysEnableICache
CyU3PSysEnableDCache
CyU3PSysEnableMMU

5.23.3.14 void CyU3PSysEnableDCache (void)

Enable the Data Cache.

Description

Function to enable the Data Cache. The MMU must also be enabled for the D-Cache to function properly. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead.

Return value

None

See Also

CyU3PSysDisableDCache
CyU3PSysEnableCacheMMU

5.23.3.15 void CyU3PSysEnableICache (void)

Enable the instruction cache.

Description

Function to enable the instruction cache. The function also sets up both caches to use random replacement strategy. This function should not be explicitly called. The CyU3PDeviceCacheControl API should be called instead. The function is available for debug purposes.

Return value

None

See Also

CyU3PSysDisableICache
CyU3PSysEnableCacheMMU

5.23.3.16 void CyU3PSysEnableMMU (void)

Enable the MMU.

Description

Function to enable the MMU. The page tables should be setup before calling this function. This function should not be explicitly called and is invoked from the library during initialization. It is provided for debug purposes.

Return value

None

See Also

CyU3PSysDisableMMU
CyU3PSysEnableCacheMMU

5.23.3.17 void CyU3PSysFlushCaches (void)

Flush both I and D Caches.

Description

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

Return value

None

5.23.3.18 void CyU3PSysFlushDCache (void)

Flush the D-Cache.

Description

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

Return value

None

5.23.3.19 void CyU3PSysFlushDRegion (uint32_t * *addr*, uint32_t *len*)

Flush a data region from the D-Cache.

Description

Function to flush a specified data region from the Data Cache. If the D-cache is enabled and the cache is handled by the application, then this API should be invoked before receiving any data using DMA.

Return value

None

Parameters

addr	Start address of the region to clean.
len	Length of the region in bytes.

5.23.3.20 void CyU3PSysFlushICache (void)

Flush the I-Cache.

Description

Function to flush the Instruction Cache.

Return value

None

5.23.3.21 void CyU3PSysFlushIRegion (uint32_t * *addr*, uint32_t *len*)

Flush a code region from the I-Cache.

Description

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

Return value

None

Parameters

addr	Start address of the region to flush.
len	Length of the region in bytes.

5.23.3.22 void CyU3PSysFlushTLBEntry (uint32_t * *addr*)

Invalidate the TLB entry corresponding to a specified MVA.

Description

This function invalidates one of the TLB entries corresponding to the specified Modified Virtual Address. Even locked down entries will be invalidated. Multiple calls of this function may be needed if multi-level page tables are in use. This function is should not be called explicitly and is provided for debug purposes.

Return value

None

Parameters

addr	Address whose TLB entry is to be invalidated.

5.23.3.23 void CyU3PSysInitTCMs (void)

Function to initialize the TCM regions.

Description

This function enables the TCMs by writing to the TCM region registers. This function should not be explicitly invoked. The library calls this function during initialization. It is provided for debug purposes.

Return value

None

5.23.3.24 void CyU3PSysLoadTLB (void)

Function to lock all valid page walks into TLB.

Description

This function locks the page table walk information for all valid addresses on the FX3 device into the TLB. This function should be called after CyU3PInitPageTable has been called. This function is should not be called explicitly and is provided for debug purposes.

Return value

None

5.23.3.25 void CyU3PSysLockTLBEntry (uint32_t * *addr*)

Function to lock the TLB entry for a page walk.

Description

This function loads the page table walk information corresponding to a specified address into the TLB and locks it. This function is should not be called explicitly and is provided for debug purposes.

Return value

None

Parameters

addr	Address range whose TLB entry is to be locked in place.

5.24 firmware/u3p_firmware/inc/cyu3os.h File Reference

This file defines the RTOS services and wrappers that are provided as part of the FX3 firmware solution. Most of these services are used by the drivers in the FX3 library and need to be provided when porting to a new OS environment.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Typedefs

- typedef struct CyU3PBytePool CyU3PBytePool
  Byte pool structure.
- typedef struct CyU3PBlockPool CyU3PBlockPool
  Block pool structure.
- typedef struct CyU3PThread CyU3PThread
  Thread structure.
- typedef void(* CyU3PThreadEntry_t)(uint32_t threadArg)
  Thread entry function type.
- typedef struct CyU3PQueue CyU3PQueue
  Message queue structure.
- typedef struct CyU3PMutex CyU3PMutex
  Mutex structure.

- typedef struct CyU3PSemaphore CyU3PSemaphore

  Semaphore structure.
- typedef struct CyU3PEvent CyU3PEvent

  Event group structure.
- typedef struct CyU3PTimer CyU3PTimer

  Timer structure.
- typedef void(∗ CyU3PTimerCb_t)(uint32_t timerArg)

  Type of callback function called on timer expiration.

Functions

- void CyU3PUndefinedHandler (void)

  The undefined instruction exception handler.
- void CyU3PPrefetchHandler (void)

  The pre-fetch error exception handler.
- void CyU3PAbortHandler (void)

  The abort error exception handler.
- void CyU3PKernelEntry (void)

  RTOS kernel entry function.
- void CyU3PApplicationDefine (void)

  The driver specific OS primitives and threads shall be created in this function.
- void CyU3POsTimerInit (uint16_t intervalMs)

  Initialize the OS scheduler timer.
- void CyU3POsTimerHandler (void)

  The OS scheduler.
- void CyU3PIrqContextSave (void)

  This function saves the active thread context when entering the IRQ context.
- void CyU3PIrqVectoredContextSave (void)

  This function saves the active thread context when entering the IRQ context.
- void CyU3PIrqContextRestore (void)

  This function restores the thread context after handling an interrupt.
- void CyU3PFiqContextSave (void)

  This function saves the active thread context when entering the FIQ context.
- void CyU3PFiqContextRestore (void)

  This function restores the thread context after handling an FIQ interrupt.
- void CyU3PIrqNestingStart (void)

  This function switches the ARM mode from IRQ to SYS to allow nesting of interrupts.
- void CyU3PIrqNestingStop (void)

  This function switches the ARM mode from SYS to IRQ at the end of a interrupt handler.
- void CyU3PMemInit (void)

  Initialize the custom heap manager for OS specific allocation calls.
- void ∗ CyU3PMemAlloc (uint32_t size)

  Allocate memory from the dynamic memory pool.
- void CyU3PMemFree (void ∗mem_p)

  Free memory allocated from the dynamic memory pool.
- void CyU3PFreeHeaps (void)

  Free up the custom heap manager and the buffer allocator.
- void CyU3PDmaBufferInit (void)

  Initialize the DMA buffer allocator.
- void CyU3PDmaBufferDeInit (void)

De-initialize the buffer allocator.

- void * CyU3PDmaBufferAlloc (uint16_t size)

  Allocate the required amount of buffer memory.
- int CyU3PDmaBufferFree (void *buffer)

  Free the previously allocated buffer area.
- void CyU3PMemSet (uint8_t *ptr, uint8_t data, uint32_t count)

  Fill a region of memory with a specified value.
- void CyU3PMemCopy (uint8_t *dest, uint8_t *src, uint32_t count)

  Copy data from one memory location to another.
- int32_t CyU3PMemCmp (const void *s1, const void *s2, uint32_t n)

  Compare the contents of two memory buffers.
- uint32_t CyU3PBytePoolCreate (CyU3PBytePool *pool_p, void *poolStart, uint32_t poolSize)

  This function creates and initializes a memory byte pool.
- uint32_t CyU3PBytePoolDestroy (CyU3PBytePool *pool_p)

  De-initialize and free up a memory byte pool.
- uint32_t CyU3PByteAlloc (CyU3PBytePool *pool_p, void **mem_p, uint32_t memSize, uint32_t waitOption)

  Allocate memory from a byte pool.
- uint32_t CyU3PByteFree (void *mem_p)

  Frees memory allocated by the CyU3PByteAlloc function.
- uint32_t CyU3PBlockPoolCreate (CyU3PBlockPool *pool_p, uint32_t blockSize, void *poolStart, uint32_t poolSize)

  Creates and initializes a memory block pool.
- uint32_t CyU3PBlockPoolDestroy (CyU3PBlockPool *pool_p)

  De-initialize a block memory pool.
- uint32_t CyU3PBlockAlloc (CyU3PBlockPool *pool_p, void **block_p, uint32_t waitOption)

  Allocate a memory buffer from a block pool.
- uint32_t CyU3PBlockFree (void *block_p)

  Frees a memory buffer allocated from a block pool.
- uint32_t CyU3PThreadCreate (CyU3PThread *thread_p, char *threadName, CyU3PThreadEntry_t entryFn, uint32_t entryInput, void *stackStart, uint32_t stackSize, uint32_t priority, uint32_t preemptThreshold, uint32_t timeSlice, uint32_t autoStart)

  This function creates a new thread.
- uint32_t CyU3PThreadDestroy (CyU3PThread *thread_ptr)

  Free up and remove a thread from the RTOS scheduler.
- CyU3PThread * CyU3PThreadIdentify (void)

  Get the thread structure corresponding to the current thread.
- uint32_t CyU3PThreadInfoGet (CyU3PThread *thread_p, uint8_t **name_p, uint32_t *priority, uint32_t *preemptionThreshold, uint32_t *timeSlice)

  Retrieve information regarding a specified thread.
- uint32_t CyU3PThreadPriorityChange (CyU3PThread *thread_p, uint32_t newPriority, uint32_t *oldPriority)

  Change the priority of a thread.
- void CyU3PThreadRelinquish (void)

  Relinquish control to the OS scheduler.
- uint32_t CyU3PThreadSleep (uint32_t timerTicks)

  Puts a thread to sleep for the specified timer ticks.
- uint32_t CyU3PThreadSuspend (CyU3PThread *thread_p)

  Suspends the specified thread.
- uint32_t CyU3PThreadResume (CyU3PThread *thread_p)

  Resume operation of a thread that was previously suspended.

- uint32_t CyU3PThreadWaitAbort (CyU3PThread *thread_p)

  Returns a thread to ready state by aborting all waits on the thread.
- uint32_t CyU3PQueueCreate (CyU3PQueue *queue_p, uint32_t messageSize, void *queueStart, uint32_t queueSize)

  Create a message queue.
- uint32_t CyU3PQueueDestroy (CyU3PQueue *queue_p)

  Free up a previously created message queue.
- uint32_t CyU3PQueueSend (CyU3PQueue *queue_p, void *src_p, uint32_t waitOption)

  Queue a new message on the specified message queue.
- uint32_t CyU3PQueuePrioritySend (CyU3PQueue *queue_p, void *src_p, uint32_t waitOption)

  Add a new message at the head of a message queue.
- uint32_t CyU3PQueueReceive (CyU3PQueue *queue_p, void *dest_p, uint32_t waitOption)

  Receive a message from a message queue.
- uint32_t CyU3PQueueFlush (CyU3PQueue *queue_p)

  Flushes all messages on a queue.
- uint32_t CyU3PMutexCreate (CyU3PMutex *mutex_p, uint32_t priorityInherit)

  Create a mutex variable.
- uint32_t CyU3PMutexDestroy (CyU3PMutex *mutex_p)

  Destroy a mutex variable.
- uint32_t CyU3PMutexGet (CyU3PMutex *mutex_p, uint32_t waitOption)

  Get the lock on a mutex variable.
- uint32_t CyU3PMutexPut (CyU3PMutex *mutex_p)

  Release the lock on a mutex variable.
- uint32_t CyU3PSemaphoreCreate (CyU3PSemaphore *semaphore_p, uint32_t initialCount)

  Create a semaphore object.
- uint32_t CyU3PSemaphoreDestroy (CyU3PSemaphore *semaphore_p)

  Destroy a semaphore object.
- uint32_t CyU3PSemaphoreGet (CyU3PSemaphore *semaphore_p, uint32_t waitOption)

  Get an instance from the specified counting semaphore.
- uint32_t CyU3PSemaphorePut (CyU3PSemaphore *semaphore_p)

  Release an instance of the specified counting semaphore.
- uint32_t CyU3PEventCreate (CyU3PEvent *event_p)

  Create an event flag group.
- uint32_t CyU3PEventDestroy (CyU3PEvent *event_p)

  Destroy an event flag group.
- uint32_t CyU3PEventSet (CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t setOption)

  Update one or more flags in an event group.
- uint32_t CyU3PEventGet (CyU3PEvent *event_p, uint32_t rqtFlag, uint32_t getOption, uint32_t *flag_p, uint32_t waitOption)

  Wait for or get the status of an event flag group.
- uint32_t CyU3PTimerCreate (CyU3PTimer *timer_p, CyU3PTimerCb_t expirationFunction, uint32_t expirationInput, uint32_t initialTicks, uint32_t rescheduleTicks, uint32_t timerOption)

  Create an application timer.
- uint32_t CyU3PTimerDestroy (CyU3PTimer *timer_p)

  Destroy an application timer object.
- uint32_t CyU3PTimerStart (CyU3PTimer *timer_p)

  Start an application timer.
- uint32_t CyU3PTimerStop (CyU3PTimer *timer_p)

  Stop operation of an application timer.
- uint32_t CyU3PTimerModify (CyU3PTimer *timer_p, uint32_t initialTicks, uint32_t rescheduleTicks)

  Modify parameters of an application timer.

- uint32_t CyU3PGetTime (void)

  Get the number of elapsed OS timer ticks since FX3 system start-up.
- void CyU3PSetTime (uint32_t newTime)

  Update the timer tick count.

5.24.1 Detailed Description

This file defines the RTOS services and wrappers that are provided as part of the FX3 firmware solution. Most of these services are used by the drivers in the FX3 library and need to be provided when porting to a new OS environment.

5.24.2 RTOS Constants

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

Description

The following constants are special parameter values passed to some OS functions to specify the desired behaviour.

CYU3P_NO_WAIT : This is a special value for waitOption which indicates that the function should not wait / block the thread and should immediately return. This is used for OS functions like CyU3PMutexGet and also for various firmware API functions which internally wait for mutexes or events.

CYU3P_WAIT_FOREVER : This is a special value for waitOption which indicates that the function should wait until the particular mutex or event has been flagged. This is used for OS functions like CyU3PMutexGet and also for various firmware API functions which internally wait for mutexes and events.

CYU3P_EVENT_AND : This is a special value for 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.

5.24.3 Exceptions and Interrupts

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

Description

The FX3 firmware generally executes in the Supervisor (SVC) mode of ARM processors, which is a privileged mode. The User and FIQ modes are currently unused by the FX3 firmware. The IRQ mode is used for the initial part of interrupt handler execution and the System mode is used for handling the second halves of long interrupts in a nested manner.

The Abort and Undefined modes are only executed when the ARM CPU encounters an execution error such as an undefined instruction, or a instruction/data access abort.

The FX3 device has an internal PL192 Vectored Interrupt Controller which is used for managing all interrupts raised by the device. The drivers for various hardware blocks in the FX3 firmware library register interrupt handlers for all interrupt sources. Event callbacks are raised to the user firmware for all relevant interrupt conditions.

Each interrupt source has a 4 bit priority value associated with it. These priority settings are unused as of now; and interrupt priority is enforced through nesting and pre-emption.

The RTOS used by the FX3 firmware allows interrupts to be nested, and this mechanism is used to allow higher priority interrupts to pre-empt lower priority ones. Thus the interrupts are classified into two groups: high priority ones that cannot be pre-empted and low priority ones that can be pre-empted.

The high priority (non pre-emptable interrupts) are:

USB interrupt

DMA interrupt for USB sockets

DMA interrupt for GPIF sockets

DMA interrupt for Serial peripheral sockets

The low priority (pre-emptable interrupts) are: System control interrupt (used for suspend/wakeup)

OS scheduler interrupt (timer based)

GPIF interrupt

SPI interrupt

I2C interrupt

I2S interrupt

UART interrupt

GPIO interrupt

The respective interrupt handlers in the drivers are responsible for enabling/disabling these interrupt sources at the appropriate time. Since disabling one or more of these interrupts at arbitrary times can cause system errors and crashes; user accessible functions to control these interrupts individually are not provided.

The FX3 SDK only provides APIs that can disable and re-enable all interrupt sources so as to ensure atomicity of critical code sections.

5.24.3.1 Exception Handler Functions

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

Description

Exceptions such as data or instruction abort, and undefined instruction may happen if the firmware image is corrupted during loading or at runtime. Since these are unexpected conditions, the FX3 firmware library does not provide any specific code to handle them. Default handlers for these conditions are provided in the cyfxtx.c file, and these can be modified by the users to match their requirements (for example, reset the FX3 device and restart operation).

5.24.4 RTOS Memory Functions

This section documents the functions that the FX3 firmware provides for dynamic memory management. These are implemented as wrappers on top of the corresponding RTOS services.

Description

The FX3 firmware makes use of a set of memory management services that are provided by the RTOS used. The firmware library also provides a set of high level dynamic memory management functions that are wrappers on top of the corresponding RTOS services.

Two flavors of memory allocation services are provided:

Byte Pool: This is a generic memory pool similar to a fixed sized heap. Memory buffers of any size can be allocated from a byte pool.

Block pool: This is a memory pool that is optimized for handling buffers 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 provides general purpose memory allocation functions that can be used to replace the standard C library memory allocation calls.

These functions can be implemented using the above mentioned byte pool and block pool functions. Implementations of these functions in source form are provided for reference, and can be modified as required by the application.

5.24.5 Thread Functions

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

Description

The FX3 firmware provides a set of thread functions that can be used by the application to create and manage threads. These functions are wrappers around the corresponding RTOS services.

The firmware framework itself consists of a number of threads that run the drivers for various peripheral blocks on the device. It is expected that these driver threads will have higher priorities than any threads created by the firmware application.

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

5.24.6 Message Queue Functions

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

Description

The FX3 firmware makes use of message queues for inter-thread data communication. A set of wrapper functions on top of the corresponding OS services are provided to allow the use of message queues from application threads.

5.24.7 Mutex functions

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

Description

The FX3 firmware provides a set of mutex functions that can be used for protection of global data structures in a multi-thread environment. These functions are all wrappers around the corresponding OS services.

5.24.8 Semaphore Functions

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

Description

In addition to mutexes used for ownership control of shared data and mutual exclusion, the FX3 firmware also provides counting semaphores that can be used for synchronization and signaling. Each semaphore is created with an initial count, and the count is decremented on each successful get operation. The get operation is failed when the count reaches zero. Each put operation on the semaphore increments the associated count.

5.24.9 Event Flag Functions

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

Description

Event flags are an advanced means for inter-thread synchronization that is provided as part of the FX3 firmware. These functions are wrappers around the corresponding functionality provided by the RTOS.

Event flag groups are groups of single bit flags or signals that can be used for inter-thread communication. Each flag in a event group can be individually signaled or waited upon by any given thread. It is possible for multiple threads to wait on different flags that are implemented by a single event group. This facility makes event flags a memory efficient means for inter-thread synchronization.

Event flags are frequently used for synchronization between various driver threads in the FX3 device.

5.24.10 Timer Functions

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

Description

Application timers are OS services that support the implementation of periodic tasks in the firmware application. Any number of application timers (subject to memory constraints) can be created by the application and the time intervals specified should be multiples of the OS timer ticks.

The OS keeps track of the registered application timers and calls the application provided callback functions every time the corresponding interval has elapsed.

5.24.11 Typedef Documentation

5.24.11.1 typedef struct CyU3PBlockPool CyU3PBlockPool

Block pool structure.

Description

A block pool is a form of heap (memory allocator) which allocated memory blocks of a fixed size selected during pool creation. This structure is more memory efficient than the more generic byte pool.

Though the block pool definition and services are part of the API library, the FX3 driver code does not make internal references to these services.

See Also

CyU3PBlockPoolCreate
CyU3PBlockPoolDestroy
CyU3PBlockAlloc
CyU3PBlockFree

5.24.11.2 typedef struct CyU3PBytePool CyU3PBytePool

Byte pool structure.

Description

A byte pool is a form of heap that supports allocation of arbitrarily sized memory blocks from a memory region specified at creation time. The total memory used by the byte pool is fixed at creation time, and allocation will fail once all of the available memory is used up.

See Also

CyU3PBytePoolCreate
CyU3PBytePoolDestroy
CyU3PByteAlloc
CyU3PByteFree

5.24.11.3 typedef struct CyU3PEvent CyU3PEvent

Event group structure.

Description

An event group is a set of event flags that an OS thread can wait for, and other threads or interrupts can notify. Each event group provides a maximum of 32 separate event flags that can be used independently. This structure is associated with each event group used in the FX3 firmware application.

See Also

CyU3PEventCreate
CyU3PEventDestroy
CyU3PEventSet
CyU3PEventGet

5.24.11.4 typedef struct CyU3PMutex CyU3PMutex

Mutex structure.

Description

This structure is associated with each Mutex object that is used in the FX3 firmware application.

See Also

CyU3PMutexCreate
CyU3PMutexDestroy
CyU3PMutexPut
CyU3PMutexGet

5.24.11.5 typedef struct CyU3PQueue CyU3PQueue

Message queue structure.

Description

Message queues are used for inter-thread communication in FX3 firmware. They are also used by the interrupt handlers for various FX3 blocks to schedule work for the driver threads to handle. This structure is associated with each message queue that is used in the firmware application.

See Also

CyU3PQueueCreate
CyU3PQueueDestroy
CyU3PQueueSend
CyU3PQueuePrioritySend
CyU3PQueueReceive
CyU3PQueueFlush

5.24.11.6 typedef struct CyU3PSemaphore CyU3PSemaphore

Semaphore structure.

Description

The semaphore service in the FX3 OS Library is a typical counting semaphore implementation. This structure is associated with each semaphore object that is used in the firmware application.

See Also

CyU3PSemaphoreCreate
CyU3PSemaphoreDestroy
CyU3PSemaphoreGet
CyU3PSemaphorePut

5.24.11.7 typedef struct CyU3PThread CyU3PThread

Thread structure.

Description

This data structure is associated with each RTOS thread that is created in the FX3 firmware. The driver modules for various FX3 blocks make use of some threads internally as well.

See Also

CyU3PThreadCreate
CyU3PThreadDestroy
CyU3PThreadIdentify
CyU3PThreadInfoGet
CyU3PThreadSleep
CyU3PThreadSuspend
CyU3PThreadResume
CyU3PThreadWaitAbort
CyU3PThreadRelinquish
CyU3PThreadPriorityChange
CyU3PThreadStackErrorNotify

5.24.11.8 typedef void(* CyU3PThreadEntry_t)(uint32_t threadArg)

Thread entry function type.

Description

This type represents the entry function for an RTOS thread created by a FX3 firmware application. The threadArg argument is a context input that is specified by the user when creating the thread.

See Also

CyU3PThread
CyU3PThreadCreate

5.24.11.9 typedef struct CyU3PTimer CyU3PTimer

Timer structure.

Description

OS timers provide a mechanism for setting up periodic tasks which will be triggered once every period number of timer ticks. This structure is associated with each OS timer object used in the FX3 firmware application.

See Also

CyU3PTimerCreate
CyU3PTimerDestroy
CyU3PTimerStart
CyU3PTimerStop
CyU3PTimerModify

5.24.11.10 typedef void(* CyU3PTimerCb_t)(uint32_t timerArg)

Type of callback function called on timer expiration.

Description

This type defines the signature of the handler callback for an OS timer object. A function of this type is associated with each timer at the time of creation and is called by the OS scheduler on a periodic basis. The timerArg parameter passed to this function is also specified at the time of creating the timer object.

See Also

CyU3PTimer
CyU3PTimerCreate

5.24.12 Function Documentation

5.24.12.1 void CyU3PAbortHandler (void)

The abort error exception handler.

Description

This function gets invoked when the ARM CPU encounters an data pre-fetch abort error. As virtual memory is not used in the system, this can only happen if data is being read from a non-existent memory location.

Return value

None

See Also

CyU3PUndefinedHandler
CyU3PPrefetchHandler

5.24.12.2 void CyU3PApplicationDefine (void)

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

Description

This function shall be implemented by the firmware library and needs to be invoked from the kernel during initialization. This is where all the threads and OS primitives for all module drivers shall be created. Though some OS calls

can be safely made at this point; scheduling, wait options and sleep functions are not expected to be functional at this point.

Return value

None

See Also

CyU3PKernelEntry

5.24.12.3 uint32_t CyU3PBlockAlloc (CyU3PBlockPool * *pool_p*, void ** *block_p*, uint32_t *waitOption*)

Allocate a memory buffer from a block pool.

Description

This function allocates a memory buffer from a block pool. The size of the buffer is fixed during the pool creation. The waitOption parameter specifies the timeout duration before the alloc call is failed.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBlockPoolCreate
CyU3PBlockFree

Parameters

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

5.24.12.4 uint32_t CyU3PBlockFree (void * *block_p*)

Frees a memory buffer allocated from a block pool.

Description

This function frees a memory buffer allocated through the CyU3PBlockAlloc call.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBlockPoolCreate
CyU3PBlockAlloc

Parameters

block_p	Pointer to buffer to be freed.

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

Creates and initializes a memory block pool.

Description

This function initializes a memory block pool from which buffers of a fixed size can be dynamically allocated. The size of the buffer is specified as a parameter to this pool create function.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBlockPoolDestroy
CyU3PBlockAlloc
CyU3PBlockFree

Parameters

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

5.24.12.6 uint32_t CyU3PBlockPoolDestroy (CyU3PBlockPool * *pool_p*)

De-initialize a block memory pool.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBlockPoolCreate

Parameters

pool_p	Handle to the block pool to be destroyed.

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

Allocate memory from a byte pool.

Description

This function is used to allocate memory buffers from a previously created memory byte pool. The waitOption specifies the timeout duration after which the memory allocation should be failed. It is permitted to use this function

from an interrupt context as long as the waitOption is set to zero or CYU3P_NO_WAIT.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBytePoolCreate
CyU3PByteFree

Parameters

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

5.24.12.8 uint32_t CyU3PByteFree (void * *mem_p*)

Frees memory allocated by the CyU3PByteAlloc function.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PBytePoolCreate
CyU3PByteAlloc

Parameters

mem_p	Pointer to memory buffer to be freed

5.24.12.9 uint32_t CyU3PBytePoolCreate (CyU3PBytePool * *pool_p*, void * *poolStart*, uint32_t *poolSize*)

This function creates and initializes a memory byte pool.

Description

This function is a wrapper around the byte pool service provided by the RTOS. It creates a fixed size general purpose heap structure that can be used for dynamic memory allocation.

Return value

CY_U3P_SUCCESS if pool was successfully initialized.

Other RTOS specific error codes in case of failure.

See Also

CyU3PBytePoolDestroy
CyU3PByteAlloc
CyU3PBlockFree

Parameters

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.
poolStart	Pointer to memory region provided for the byte pool. This address needs to be 16 byte aligned.
poolSize	Size of the memory region provided for the byte pool. This size needs to be a multiple of 16 bytes.

5.24.12.10 uint32_t CyU3PBytePoolDestroy (CyU3PBytePool * *pool_p*)

De-initialize and free up a memory byte pool.

Description

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

Return value

CY_U3P_SUCCESS on success.

Other RTOS error codes on failure.

See Also

CyU3PBytePoolCreate

Parameters

pool_p	Handle to the byte pool to be freed up

5.24.12.11 void* CyU3PDmaBufferAlloc (uint16_t *size*)

Allocate the required amount of buffer memory.

Description

This function allocates buffers of the required size from the memory region reserved through the CyU3PDmaBuffer-Init function. It is called by the DMA APIs when creating DMA channels, and can also be used directly by the user application code.

This implementation is expected to ensure that the buffers allocated are aligned to data cache lines.

Return value

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

See Also

CyU3PDmaBufferInit
CyU3PDmaBufferDeInit
CyU3PDmaBufferFree

Parameters

size	The size in bytes of the buffer required.

5.24.12.12 void CyU3PDmaBufferDeInit (void)

De-initialize the buffer allocator.

Description

This function de-initializes the DMA buffer manager. The function shall be invoked on DMA module de-init. This is provided in source format so that it can be modified as per user requirement. This function must not be called directly by the user application.

Return value

None

See Also

CyU3PDmaBufferInit

5.24.12.13 int CyU3PDmaBufferFree (void * *buffer*)

Free the previously allocated buffer area.

Description

Frees up a buffer previously allocated through the CyU3PDmaBufferAlloc function. This function is used internally by the DMA API when freeing up DMA channels, and can also be used directly by the user application code.

Return value

0 if the buffer is freed successfully.

Non-zero value otherwise.

See Also

CyU3PDmaBufferInit
CyU3PDmaBufferDeInit
CyU3PDmaBufferAlloc

Parameters

buffer	Address of buffer to be freed.

5.24.12.14 void CyU3PDmaBufferInit (void)

Initialize the DMA buffer allocator.

Description

This function creates an allocator that provides the DMA buffers required for data transfers through the FX3 device. The DMA buffers are also allocated from a pre-assigned memory region within the FX3 RAM.

This allocator needs to ensure that all buffers managed by it are aligned to data cache lines. i.e., That all buffers start at addresses that are multiples of 32 bytes and span a multiple of 32 bytes.

This function is called internally by the FX3 firmware library and should not be called directly by the user application.

Return value

None

See Also

CyU3PDmaBufferDeInit
CyU3PDmaBufferAlloc
CyU3PDmaBufferFree

5.24.12.15 uint32_t CyU3PEventCreate (CyU3PEvent * *event_p*)

Create an event flag group.

Description

This function creates an event flag group consisting of 32 independent event flags. The application is free to use as many flags as required from this group. If more than 32 flags are required, multiple event flag groups have to be created.

As with other OS service create functions, the caller is expected to allocate memory for the Event data structure.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PEventDestroy
CyU3PEventSet
CyU3PEventGet

Parameters

event_p	Pointer to event structure to be initialized.

5.24.12.16 uint32_t CyU3PEventDestroy (CyU3PEvent * *event_p*)

Destroy an event flag group.

Description

This function frees up an event flag group. Any threads waiting on one or more flags in the group will be re-activated and will receive an error code from the event wait function.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PEventCreate

Parameters

event_p	Pointer to event group to be destroyed.

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

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

Description

This function is used to the get the current status of the flags in a event group. This can also be used to wait until one or more flags of interest have been signaled. The maximum amount of time to wait is specified through the waitOption parameter.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PEventSet

Parameters

event_p	Pointer to event group to be queried.
rqtFlag	Bit-mask that selects event flags of interest.
getOption	Type of operation to perform: CYU3P_EVENT_AND: Use to wait until all selected flags are signaled. CYU3P_EVENT_AND_CLEAR: Same as above, and clear the flags on read. CYU3P_EVENT_OR: Wait until any of selected flags are signaled. CYU3P_EVENT_OR_CLEAR: Same as above, and clear the flags on read.
flag_p	Output parameter filled in with the state of the flags.
waitOption	Timeout duration in timer ticks.

5.24.12.18 uint32_t CyU3PEventSet (CyU3PEvent * *event_p*, uint32_t *rqtFlag*, uint32_t *setOption*)

Update one or more flags in an event group.

Description

This function is used to set or clear one or more flags that are part of a specified event group. The parameters can be selected so as to set a number of specified flags, or to clear of number of specified flags.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PEventGet

Parameters

event_p	Pointer to event group to update.
rqtFlag	Bit-mask that selects event flags of interest.
setOption	Type of set operation to perform: CYU3P_EVENT_AND: The rqtFlag will be ANDed with the current flags. CYU3P_EVENT_OR: The rqtFlag will be ORed with the current flags.

5.24.12.19 void CyU3PFiqContextRestore (void)

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

Description

This function is provided as part of the OS and restores thread state saved at the beginning of the FIQ handler using the CyU3PFiqContextSave API.

Return value

None

See Also

CyU3PFiqContextSave

5.24.12.20 void CyU3PFiqContextSave (void)

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

Description

This function is provided as part of the OS and saves thread state when entering a Fast Interrupt (FIQ) handler routine. As the FX3 firmware library currently does not use a FIQ, this function is not used.

Return value

None

See Also

CyU3PFiqContextRestore

5.24.12.21 void CyU3PFreeHeaps (void)

Free up the custom heap manager and the buffer allocator.

Description

This function is called in preparation to a reset of the FX3 device and is intended to allow the user to free up the custom heap manager and the buffer allocator that are created in the CyU3PMemInit and the CyU3PDmaBufferInit functions respectively.

Return value

None

See Also

CyU3PMemInit
CyU3PDmaBufferInit

5.24.12.22 uint32_t CyU3PGetTime (void)

Get the number of elapsed OS timer ticks since FX3 system start-up.

Description

The function returns the number of elapsed OS timer ticks since the FX3 OS was started up.

Return value

Current OS timer tick count.

See Also

CyU3PSetTime

5.24.12.23 void CyU3PIrqContextRestore (void)

This function restores the thread context after handling an interrupt.

Description

This function is provided as part of the OS and allows the previously backed up thread state to be restored at the end of a IRQ handler call. The same function is used to restore thread state saved by the CyU3PIrqContextSave and CyU3PIrqVectoredContextSave functions.

Return value

None

See Also

CyU3PIrqContextSave
CyU3PIrqVectoredContextSave

5.24.12.24 void CyU3PIrqContextSave (void)

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

Description

This function is provided as part of the OS to allow any thread state to be backed up on entry to a IRQ handler routine. This function is only applicable when the VIC is not being used, and is generally not used in FX3 firmware.

See the CyU3PIrqVectoredContextSave function for the equivalent function used in FX3 firmware.

Return value

None

See Also

CyU3PIrqContextRestore
CyU3PIrqVectoredContextSave
CyU3PFiqContextSave

5.24.12.25 void CyU3PIrqNestingStart (void)

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

Description

Nesting of interrupts on an ARMv5 controller requires that the current interrupt handler switch the ARM execution mode to SYS mode. This function is used to do this switching. In case interrupt nesting is not required for this platform, this function can be a No-Operation.

Return value

None

See Also

CyU3PIrqNestingStop

5.24.12.26 void CyU3PIrqNestingStop (void)

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

Description

Nesting of interrupts on an ARMv5 controller requires that the ARM execution mode be switched to SYS mode at the beginning of the handler and back to IRQ mode at the end. This function is used to do the switch the mode back to IRQ at the end of an interrupt handler. In case interrupt nesting is not required for this platform, this function can be a No-operation.

Return value

None

See Also

CyU3PIrqNestingStart

5.24.12.27 void CyU3PIrqVectoredContextSave (void)

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

Description

This function is provided as part of the OS to allow any thread state to be backed up on entry to a IRQ handler routine. This function is applicable when the device in use has a Vectored Interrupt Controller (VIC) and is used by all interrupt handlers in the FX3 firmware.

Return value

None

See Also

CyU3PIrqContextRestore
CyU3PIrqContextSave
CyU3PFiqContextSave

5.24.12.28 void CyU3PKernelEntry (void)

RTOS kernel entry function.

Description

The function call is expected to initialize the RTOS. This function needs to be invoked as the last function from the main () function. It is expected that the firmware shall always function in the SVC mode.

Return value

None - The function does not return at all.

See Also

CyU3PApplicationDefine

5.24.12.29 void* CyU3PMemAlloc (uint32_t *size*)

Allocate memory from the dynamic memory pool.

Description

This function is the malloc equivalent for allocating memory from the pool created by the CyU3PMemInit function. This function needs to be implemented as part of the RTOS porting to the FX3 device. This function needs to be capable of allocating memory even if called from an interrupt handler.

Return value

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

See Also

CyU3PMemInit
CyU3PMemFree

Parameters

size	The size in bytes of the buffer to be allocated.

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

Compare the contents of two memory buffers.

Description

This is a memcmp equivalent function that does a byte by byte comparison of two memory buffers.

Return value

0 if the two buffers hold the same data.

Negative if the data in s1 has a lower ASCII value than that in s2.

Positive if the data in s1 has a higher ASCII value that that in s2.

Parameters

s1	Pointer to first memory buffer.
s2	Pointer to second memory buffer.
n	Size in bytes of the buffers to compare.

5.24.12.31 void CyU3PMemCopy (uint8_t * *dest*, uint8_t * *src*, uint32_t *count*)

Copy data from one memory location to another.

Description

This is a memcpy equivalent function that is provided and used by the firmware library. The implementation does not handle the case of overlapping buffers.

Return value

None

Parameters

dest	Pointer to destination buffer.
src	Pointer to source buffer.
count	Size of the buffer to be copied.

5.24.12.32 void CyU3PMemFree (void * *mem_p*)

Free memory allocated from the dynamic memory pool.

Description

This function is the free equivalent that frees memory allocated through the CyU3PMemAlloc function. This function is also expected to be able to free memory in an interrupt context.

Return value

None

See Also

CyU3PMemInit
CyU3PMemAlloc

Parameters

mem_p	Pointer to memory buffer to be freed.

5.24.12.33 void CyU3PMemInit (void)

Initialize the custom heap manager for OS specific allocation calls.

Description

This function creates a memory pool that can be used in place of the system heap. All dynamic memory allocation for OS data structures, thread stacks, and firmware data buffers should be allocated out of this memory pool. The size and location of this memory pool needs to be adjusted as per the user requirements by modifying this function. This function is called internally by the FX3 library, and should not be called directly by the user code.

Return value

None

See Also

CyU3PMemAlloc
CyU3PMemFree
CyU3PFreeHeaps

5.24.12.34 void CyU3PMemSet (uint8_t * *ptr*, uint8_t *data*, uint32_t *count*)

Fill a region of memory with a specified value.

Description

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

Return value

None

Parameters

ptr	Pointer to the buffer to be filled.
data	Value to fill the buffer with.
count	Size of the buffer in bytes.

5.24.12.35 uint32_t CyU3PMutexCreate (CyU3PMutex * *mutex_p*, uint32_t *priorityInherit*)

Create a mutex variable.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PMutexDestroy
CyU3PMutexGet
CyU3PMutexPut

Parameters

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

5.24.12.36 uint32_t CyU3PMutexDestroy (CyU3PMutex * *mutex_p*)

Destroy a mutex variable.

Description

This function destroys a mutex that was created using the CyU3PMutexCreate API.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PMutexCreate

Parameters

mutex_p	Pointer to mutex to be destroyed.

5.24.12.37 uint32_t CyU3PMutexGet (CyU3PMutex * *mutex_p*, uint32_t *waitOption*)

Get the lock on a mutex variable.

Description

This function is used to wait on a mutex variable and to get a lock on it. The maximum amount of time to wait is specified through the waitOption parameter.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PMutexCreate
CyU3PMutexDestroy

Parameters

mutex_p	Pointer to mutex to be acquired.
waitOption	Timeout duration in timer ticks.

5.24.12.38 uint32_t CyU3PMutexPut (CyU3PMutex * *mutex_p*)

Release the lock on a mutex variable.

Description

This function is used to release the lock on a mutex variable. The mutex can then be acquired by the highest priority thread from among those waiting for it.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PMutexGet

Parameters

mutex_p	Pointer to mutex to be released.

5.24.12.39 void CyU3POsTimerHandler (void)

The OS scheduler.

Description

This function is implemented by the RTOS kernel and is called from the OS timer interrupt. This function call notifies the kernel scheduler that another time tick has elapsed.

Return value

None

5.24.12.40 void CyU3POsTimerInit (uint16_t *intervalMs*)

Initialize the OS scheduler timer.

Description

This function is implemented by the firmware library and is invoked from the RTOS kernel during initialization.

The OS_TIMER_TICK shall be defined by this function and all wait durations are calculated based on this. By default, the OS timer tick is configured to be 1ms. It is recommended that this default tick duration be retained to guarantee fast and accurate system response.

If the timer tick needs to be modified, this API can be invoked only after the RTOS has been initialized. This function can support a timer tick range from 1ms to 1000ms. Any other value of the timer tick will be discarded and the timer tick will be set to 1ms.

The clock for this timer is derived from the 32KHz standby clock. The actual tick interval can have an error of upto +/- 4%. The time interval will be accurate only as long as the interrupts are running.

Return value

None

Parameters

intervalMs	OS Timer tick interval in millisecond.

5.24.12.41 void CyU3PPrefetchHandler (void)

The pre-fetch error exception handler.

Description

This function gets invoked when the ARM CPU encounters an instruction pre-fetch error. Since virtual memory is not used in the system, this can only happen if the device is trying to fetch instructions from non-existent memory.

Return value

None

See Also

CyU3PUndefinedHandler
CyU3PAbortHandler

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

Create a message queue.

Description

Create a message queue that can hold a specified number of messages of a specified size. The memory for the queue should be allocated and passed in by the caller.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueueDestroy
CyU3PQueueSend
CyU3PQueuePrioritySend
CyU3PQueueReceive
CyU3PQueueFlush

Parameters

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

5.24.12.43 uint32_t CyU3PQueueDestroy (CyU3PQueue * *queue_p*)

Free up a previously created message queue.

Description

This function frees up a previously created message queue. Any function call that is waiting to send or receive messages on this queue will return with an error.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueueCreate

Parameters

queue_p	Pointer to the queue to be destroyed.

5.24.12.44 uint32_t CyU3PQueueFlush (CyU3PQueue * *queue_p*)

Flushes all messages on a queue.

Description

The function removes all pending messages on the specified queue.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueueSend
CyU3PQueuePrioritySend
CyU3PQueueReceive

Parameters

queue_p	Pointer to the queue to be flushed.

5.24.12.45 uint32_t CyU3PQueuePrioritySend (CyU3PQueue * *queue_p*, void * *src_p*, uint32_t *waitOption*)

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

Description

This function is used to send a high priority message to a message queue. This message will be placed at the head of the queue instead of at the tail. As in the case of the CyU3PQueueSend API, this waits for the specified duration or until a message slot is free in the queue.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueueSend
CyU3PQueueReceive
CyU3PQueueFlush

Parameters

queue_p	Pointer to the message queue structure.
src_p	Pointer to message buffer.
waitOption	Timeout duration in timer ticks.

5.24.12.46 uint32_t CyU3PQueueReceive (CyU3PQueue * *queue_p*, void * *dest_p*, uint32_t *waitOption*)

Receive a message from a message queue.

Description

This function receives the message at the head of the specified queue. If the queue is empty, it waits until the specified timeout period has elapsed, or until the queue is non-empty. If calling from an interrupt handler, the waitOption parameter needs to be set to CYU3P_NO_WAIT.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueueSend
CyU3PQueuePrioritySend
CyU3PQueueFlush

Parameters

queue_p	Pointer to the message queue.
dest_p	Pointer to buffer where the message should be copied.
waitOption	Timeout duration in timer ticks.

5.24.12.47 uint32_t CyU3PQueueSend (CyU3PQueue * *queue_p*, void * *src_p*, uint32_t *waitOption*)

Queue a new message on the specified message queue.

Description

This function adds a new message on to the specified message queue. If the queue is full, it waits for a message slot to be freed. The amount of time to wait is specified as a parameter. In case this function is called from an interrupt handler, the time-out should be specified as CYU3P_NO_WAIT.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PQueuePrioritySend
CyU3PQueueReceive
CyU3PQueueFlush

Parameters

queue_p	Queue to add a new message to.
src_p	Pointer to buffer containing message.
waitOption	Timeout value to wait on the queue.

5.24.12.48 uint32_t CyU3PSemaphoreCreate (CyU3PSemaphore * *semaphore_p*, uint32_t *initialCount*)

Create a semaphore object.

Description

This function creates a semaphore object with the specified initial count. The semaphore data structure has to be allocated by the caller and will be initialized by this function call.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PSemaphoreDestroy
CyU3PSemaphoreGet
CyU3PSemaphorePut

Parameters

semaphore_p	Pointer to semaphore to be initialized.
initialCount	Initial count to associate with semaphore.

5.24.12.49 uint32_t CyU3PSemaphoreDestroy (CyU3PSemaphore * *semaphore_p*)

Destroy a semaphore object.

Description

This function destroys a semaphore object. All threads waiting to get the semaphore will receive an error code identifying that the object has been removed.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PSemaphoreCreate

Parameters

semaphore_p	Pointer to semaphore to be destroyed.

5.24.12.50 uint32_t CyU3PSemaphoreGet (CyU3PSemaphore * *semaphore_p*, uint32_t *waitOption*)

Get an instance from the specified counting semaphore.

Description

This function is used to get an instance (i.e., decrement the count by one) from the specified counting semaphore. If the count is already zero, the function will wait until the count becomes non-zero. The maximum interval to wait for is specified using the waitOption parameter.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PSemaphorePut

Parameters

semaphore_p	Pointer to semaphore to get.
waitOption	Timeout duration in timer ticks.

5.24.12.51 uint32_t CyU3PSemaphorePut (CyU3PSemaphore * *semaphore_p*)

Release an instance of the specified counting semaphore.

Description

This function releases an instance (increments the count by one) of the specified counting semaphore. The semaphore put can be done from any thread and does not need to be done by the same thread as called the get function.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PSemaphoreCreate
CyU3PSemaphoreDestroy
CyU3PSemaphoreGet

Parameters

semaphore_p	Pointer to semaphore to put.

5.24.12.52 void CyU3PSetTime (uint32_t *newTime*)

Update the timer tick count.

Description

This function modifies the timer tick count that is started at system reset. This function should ideally not be called after the application is started up, as it can affect the operation of timers and threads.

Return value

None

See Also

CyU3PGetTime

Parameters

newTime	New timer tick value to set.

5.24.12.53 uint32_t CyU3PThreadCreate (CyU3PThread * *thread_p*, char * *threadName*, CyU3PThreadEntry_t *entryFn*, uint32_t *entryInput*, void * *stackStart*, uint32_t *stackSize*, uint32_t *priority*, uint32_t *preemptThreshold*, uint32_t *timeSlice*, uint32_t *autoStart*)

This function creates a new thread.

Description

This function creates a new application thread with the specified parameters. This function call must be made only after the RTOS kernel has been started.

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadDestroy
CyU3PThreadIdentify
CyU3PThreadInfoGet
CyU3PThreadPriorityChange
CyU3PThreadRelinquish
CyU3PThreadSleep
CyU3PThreadSuspend
CyU3PThreadResume
CyU3PThreadWaitAbort
CyU3PThreadStackErrorNotify

Parameters

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

5.24.12.54 uint32_t CyU3PThreadDestroy (CyU3PThread * *thread_ptr*)

Free up and remove a thread from the RTOS scheduler.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate

Parameters

thread_ptr	Pointer to the thread structure.

5.24.12.55 CyU3PThread* CyU3PThreadIdentify (void)

Get the thread structure corresponding to the current thread.

Description

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

Return value

Pointer to the thread structure of the currently running thread.

See Also

CyU3PThreadCreate
CyU3PThreadInfoGet

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

Retrieve information regarding a specified thread.

Description

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

Valid (non-NULL) pointers need to be provided only for the output fields that need to be retrieved.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate
CyU3PThreadIdentify

Parameters

thread_p	Pointer to thread to be queried.
name_p	Output variable to be filled with the thread's name string.
priority	Output variable to be filled with the thread priority.
preemption-Threshold	Output variable to be filled with the pre-emption threshold.
timeSlice	Output variable to be filled with the time slice value.

5.24.12.57 uint32_t CyU3PThreadPriorityChange (CyU3PThread * *thread_p*, uint32_t *newPriority*, uint32_t * *oldPriority*)

Change the priority of a thread.

Description

This function can be used to change the priority of a specified thread. Though this is not expected to be used

commonly, it can be used for temporary priority changes to prevent deadlocks.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate
CyU3PThreadRelinquish
CyU3PThreadSleep

Parameters

thread_p	Pointer to thread to be modified.
newPriority	Priority value to assign to the thread.
oldPriority	Output variable that will hold the original priority.

5.24.12.58 void CyU3PThreadRelinquish (void)

Relinquish control to the OS scheduler.

Description

This is a RTOS call for fair scheduling which relinquishes control to other ready threads that are at the same priority level. The thread that relinquishes control remains in ready state and can regain control if there are no other ready threads with the same priority level.

Return value

None

See Also

CyU3PThreadCreate
CyU3PThreadSleep
CyU3PThreadSuspend

5.24.12.59 uint32_t CyU3PThreadResume (CyU3PThread * *thread_p*)

Resume operation of a thread that was previously suspended.

Description

This function is used to resume operation of a thread that was suspended using the CyU3PThreadSuspend call. Threads that are suspended because they are blocked on Mutexes or Events cannot be resumed using this call.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadSuspend

Parameters

thread_p	Pointer to thread to be resumed.

5.24.12.60 uint32_t CyU3PThreadSleep (uint32_t *timerTicks*)

Puts a thread to sleep for the specified timer ticks.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate
CyU3PThreadRelinquish
CyU3PThreadSuspend

Parameters

timerTicks	Number of timer ticks to sleep.

5.24.12.61 uint32_t CyU3PThreadSuspend (CyU3PThread * *thread_p*)

Suspends the specified thread.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate
CyU3PThreadRelinquish
CyU3PThreadSleep
CyU3PThreadResume

Parameters

thread_p	Pointer to the thread to be suspended.

5.24.12.62 uint32_t CyU3PThreadWaitAbort (CyU3PThread * *thread_p*)

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

Description

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.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PThreadCreate
CyU3PThreadSleep
CyU3PThreadSuspend
CyU3PThreadResume

Parameters

thread_p	Pointer to the thread to restore.

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

Create an application timer.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PTimerCb_t
CyU3PTimerDestroy
CyU3PTimerStart
CyU3PTimerStop
CyU3PTimerModify
CyU3PGetTime
CyU3PSetTime

Parameters

timer_p	Pointer to the timer structure to be initialized.
expiration-Function	Pointer to callback function called on timer expiration.
expirationInput	Parameter to be passed to the callback function.
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.
rescheduleTicks	The reload value for the timer. If set to zero, the timer will be a one-shot timer.
timerOption	Timer start control: CYU3P_AUTO_ACTIVATE: Start the timer immediately. CYU3P_NO_A-CTIVATE: Timer needs to be started explicitly.

5.24.12.64 uint32_t CyU3PTimerDestroy (CyU3PTimer * *timer_p*)

Destroy an application timer object.

Description

This function destroys and application timer object.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PTimerCreate

Parameters

timer_p	Pointer to the timer to be destroyed.

5.24.12.65 uint32_t CyU3PTimerModify (CyU3PTimer * *timer_p*, uint32_t *initialTicks*, uint32_t *rescheduleTicks*)

Modify parameters of an application timer.

Description

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

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PTimerCreate
CyU3PTimerStart
CyU3PTimerStop

Parameters

timer_p	Pointer to the timer.
initialTicks	Initial count to be set for the timer.
rescheduleTicks	Reload count for the timer.

5.24.12.66 uint32_t CyU3PTimerStart (CyU3PTimer * *timer_p*)

Start an application timer.

Description

This function activates a previously stopped timer. This operation can be used to start a timer that was created with the CYU3P_NO_ACTIVATE option, or to re-start a one-shot or continuous timer that was stopped previously.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PTimerCreate
CyU3PTimerStop
CyU3PTimerModify

Parameters

timer_p	Timer to be started.

5.24.12.67 uint32_t CyU3PTimerStop (CyU3PTimer * *timer_p*)

Stop operation of an application timer.

Description

This function can be used to stop operation of an application timer. The parameters associated with the timer can then be modified using the CyU3PTimerModify call.

Return value

CY_U3P_SUCCESS (0) on success.

Other error codes on failure.

See Also

CyU3PTimerStart
CyU3PTimerModify

Parameters

timer_p	Pointer to timer to be stopped.

5.24.12.68 void CyU3PUndefinedHandler (void)

The undefined instruction exception handler.

Description

This function gets invoked when the ARM CPU encounters an undefined instruction. This happens when the firmware loses control and jumps to unknown locations. This should not occur as a part of normal operation sequence, and may be seen in case of memory corruption.

Return value

None

See Also

CyU3PPrefetchHandler
CyU3PAbortHandler

5.25 firmware/u3p_firmware/inc/cyu3pib.h File Reference

The Processor Interface Block (PIB) on the FX3 device contains the GPIF-II controller and the associated DMA sockets and configuration registers. The PIB driver in FX3 firmware is responsible for managing PIB bound data transfers and interrupts. This file contains the data types and API definitions for the general P-port functionality that is independent of the electrical protocol implemented by GPIF-II.

```
#include "cyu3os.h"
#include "cyu3types.h"
#include "cyu3system.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PPibClock_t

  Clock configuration information for the PIB block.

Macros

- #define CYU3P_GET_PIB_ERROR_TYPE(param) ((CyU3PPibErrorType)((param) & 0x3F))

  Get the PIB error code from the CYU3P_PIB_INTR_ERROR callback argument.
- #define CYU3P_GET_GPIF_ERROR_TYPE(param) ((CyU3PGpifErrorType)((param) & 0x7C00))

  Get the GPIF error code from the CYU3P_PIB_INTR_ERROR callback argument.

Typedefs

- typedef struct CyU3PPibClock_t CyU3PPibClock_t

  Clock configuration information for the PIB block.
- typedef enum CyU3PPibErrorType CyU3PPibErrorType

  Enumeration of P-port error types.
- typedef enum CyU3PGpifErrorType CyU3PGpifErrorType

  Enumeration of GPIF error types.
- typedef enum CyU3PPibIntrType CyU3PPibIntrType

  Enumeration of P-port interrupt event types.
- typedef void(∗ CyU3PPibIntrCb_t)(CyU3PPibIntrType cbType, uint16_t cbArg)

  Type of callback function that will be called on receiving a generic P-port interrupt.
- typedef enum CyU3PPmmcState CyU3PPmmcState

  List of MMC-Slave states.
- typedef enum CyU3PPmmcEventType CyU3PPmmcEventType

  List of MMC-Slave Event notifications.
- typedef void(∗ CyU3PPmmcIntrCb_t)(CyU3PPmmcEventType cbType, uint32_t cbArg)

  Callback function type used for MMC-Slave event notifications.

Enumerations

- enum 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 }

  Enumeration of P-port error types.
- 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 }

  Enumeration of GPIF error types.

- enum CyU3PPibIntrType { CYU3P_PIB_INTR_DLL_UPDATE = 1, CYU3P_PIB_INTR_PPCONFIG = 2, CYU3P_PIB_INTR_ERROR = 4 }

  Enumeration of P-port interrupt event types.

- enum CyU3PPmmcState { CY_U3P_PMMC_IDLE = 0, CY_U3P_PMMC_READY, CY_U3P_PMMC_IDENTIFICATION, CY_U3P_PMMC_STANDBY, CY_U3P_PMMC_TRANS, CY_U3P_PMMC_SENDDATA, CY_U3P_PMMC_RECVDATA, CY_U3P_PMMC_PGM, CY_U3P_PMMC_DISCONNECT, CY_U3P_PMMC_BUSTEST, CY_U3P_PMMC_SLEEP, CY_U3P_PMMC_WAITIRQ, CY_U3P_PMMC_INACTIVE, CY_U3P_PMMC_BOOT, CY_U3P_PMMC_PREIDLE, CY_U3P_PMMC_PREBOOT }

  List of MMC-Slave states.

- enum CyU3PPmmcEventType { CYU3P_PMMC_GOIDLE_CMD = 0, CYU3P_PMMC_CMD5_SLEEP, CYU3P_PMMC_CMD5_AWAKE, CYU3P_PMMC_CMD6_SWITCH, CYU3P_PMMC_CMD12_STOP, CYU3P_PMMC_CMD15_INACTIVE, CYU3P_PMMC_DIRECT_WRITE, CYU3P_PMMC_DIRECT_READ, CYU3P_PMMC_SOCKET_NOT_READY, CYU3P_PMMC_CMD7_SELECT }

  List of MMC-Slave Event notifications.

Functions

- CyU3PReturnStatus_t CyU3PPibSelectMmcSlaveMode (void)

  Select the MMC Slave mode of operation for the P-port interface block.

- CyU3PReturnStatus_t CyU3PPibInit (CyBool_t doInit, CyU3PPibClock_t *pibClock)

  Initialize the P-port interface block.

- CyU3PReturnStatus_t CyU3PPibDeInit (void)

  De-initialize the P-port interface block.

- CyU3PReturnStatus_t CyU3PSetPportDriveStrength (CyU3PDriveStrengthState_t pportDriveStrength)

  Set the IO drive strength for the Pport.

- void CyU3PPibRegisterCallback (CyU3PPibIntrCb_t cbFunc, uint32_t intMask)

  Register a callback function for notification of PIB interrupts.

- CyU3PReturnStatus_t CyU3PPibSetInterruptPriority (CyBool_t isHigh)

  Set the priority level for PIB/GPIF interrupts in the FX3 system.

- void CyU3PPibSelectIntSources (CyBool_t pibSockEn, CyBool_t gpifIntEn, CyBool_t pibErrEn, CyBool_t mboxIntEn, CyBool_t wakeupEn)

  Select the sources that trigger an FX3 interrupt to external processor.

- void CyU3PPmmcRegisterCallback (CyU3PPmmcIntrCb_t cbFunc)

  Register a callback for notification of MMC-Slave (PMMC) events.

- CyU3PReturnStatus_t CyU3PPmmcEnableDirectAccess (CyBool_t enable, uint8_t wrSock, uint8_t rdSock)

  Enable MMC direct read/write functionality.

5.25.1 Detailed Description

The Processor Interface Block (PIB) on the FX3 device contains the GPIF-II controller and the associated DMA sockets and configuration registers. The PIB driver in FX3 firmware is responsible for managing PIB bound data transfers and interrupts. This file contains the data types and API definitions for the general P-port functionality that is independent of the electrical protocol implemented by GPIF-II.

5.25.2 Macro Definition Documentation

5.25.2.1 #define CYU3P_GET_GPIF_ERROR_TYPE(*param*) ((CyU3PGpifErrorType)((param) & 0x7C00))

Get the GPIF error code from the CYU3P_PIB_INTR_ERROR callback argument.

Description

This macro is used to get the GPIF error code part from the cbArg parameter passed to the PIB callback as part of a CYU3P_PIB_INTR_ERROR event.

See Also

CyU3PGpifErrorType

5.25.2.2 #define CYU3P_GET_PIB_ERROR_TYPE(*param*) ((CyU3PPibErrorType)((param) & 0x3F))

Get the PIB error code from the CYU3P_PIB_INTR_ERROR callback argument.

Description

This macro is used to get the PIB error code part from the cbArg parameter passed to the PIB callback as part of a CYU3P_PIB_INTR_ERROR event.

See Also

CyU3PPibErrorType

5.25.3 Typedef Documentation

5.25.3.1 typedef enum CyU3PGpifErrorType CyU3PGpifErrorType

Enumeration of GPIF error types.

Description

In addition to the PIB level errors, there can be cases where GPIF state machine specific errors occur during G-PIF operation. The cbArg parameter passed to the callback in the case of a CYU3P_PIB_INTR_ERROR event is composed of a PIB error code as well as a GPIF error code. This enumerated type lists the various GPIF specific error types that are defined for the FX3 device.

The CYU3P_GET_GPIF_ERROR_TYPE macro can be used to get the GPIF error code from the cbArg parameter.

See Also

CyU3PPibErrorType
CYU3P_GET_GPIF_ERROR_TYPE

5.25.3.2 typedef struct CyU3PPibClock_t CyU3PPibClock_t

Clock configuration information for the PIB block.

Description

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

The default values used by the driver are:

clkDiv = 2, isHalfDiv = CyFalse, isDllEnable = CyFalse, and clkSrc = CY_U3P_SYS_CLK.

See Also

CyU3PSysClockSrc_t
CyU3PPibInit

5.25.3.3 typedef enum CyU3PPibErrorType CyU3PPibErrorType

Enumeration of P-port error types.

Description

The P-port interface block (PIB) of the FX3 device can encounter various errors during data transfers performed across the GPIF interface. This enumerated type lists the PIB level error conditions that are notified to the user application through the CYU3P_PIB_INTR_ERROR interrupt callback. The cbArg parameter passed to the callback will indicate the PIB error code as well as the GPIF error code.

The CYU3P_GET_PIB_ERROR_TYPE macro can be used to decode the PIB error type from the callback argument.

See Also

CyU3PPibIntrType
CyU3PPibRegisterCallback
CyU3PGpifErrorType
CYU3P_GET_PIB_ERROR_TYPE

5.25.3.4 typedef void(* CyU3PPibIntrCb_t)(CyU3PPibIntrType cbType,uint16_t cbArg)

Type of callback function that will be called on receiving a generic P-port interrupt.

Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. This function is used to register a callback function that will be invoked when one of these interrupts is triggered.

See Also

CyU3PPibRegisterCallback

5.25.3.5 typedef enum CyU3PPibIntrType CyU3PPibIntrType

Enumeration of P-port interrupt event types.

Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. These interrupts indicate events such as error conditions, mailbox message reception and Interface Config register updates. This enumerated type lists the various interrupt events that are valid for the PIB interrupt.

See Also

CyU3PPibIntrCb_t
CyU3PPibRegisterCallback

5.25.3.6 typedef enum CyU3PPmmcEventType CyU3PPmmcEventType

List of MMC-Slave Event notifications.

Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

See Also

CyU3PPmmcIntrCb_t

5.25.3.7 typedef void(∗ CyU3PPmmcIntrCb_t)(CyU3PPmmcEventType cbType,uint32_t cbArg)

Callback function type used for MMC-Slave event notifications.

Description

This function pointer type defines the signature of the callback function that will be called to notify the user of MMC-Slave (PMMC) Mode events.

See Also

CyU3PPmmcEventType
CyU3PPmmcRegisterCallback

5.25.3.8 typedef enum CyU3PPmmcState CyU3PPmmcState

List of MMC-Slave states.

Description

When the FX3 is configured as an MMC-Slave device, the MMC interface goes through a set of states (as defined in the MMC specification) in response to commands issued by the MMC host. This type lists the various MMC Slave states for the FX3 device.

5.25.4 Enumeration Type Documentation

5.25.4.1 enum CyU3PGpifErrorType

Enumeration of GPIF error types.

Description

In addition to the PIB level errors, there can be cases where GPIF state machine specific errors occur during G-PIF operation. The cbArg parameter passed to the callback in the case of a CYU3P_PIB_INTR_ERROR event is composed of a PIB error code as well as a GPIF error code. This enumerated type lists the various GPIF specific error types that are defined for the FX3 device.

The CYU3P_GET_GPIF_ERROR_TYPE macro can be used to get the GPIF error code from the cbArg parameter.

See Also

CyU3PPibErrorType
CYU3P_GET_GPIF_ERROR_TYPE

Enumerator

CYU3P_GPIF_ERR_NONE No GPIF state machine errors.

CYU3P_GPIF_ERR_INADDR_OVERWRITE Content of INGRESS_ADDR register is overwritten before read.

CYU3P_GPIF_ERR_EGADDR_INVALID Attempt to read EGRESS_ADDR register before it is written to.

CYU3P_GPIF_ERR_DATA_READ_ERR Read from DMA data thread which is not ready.

CYU3P_GPIF_ERR_DATA_WRITE_ERR Write to DMA data thread which is not ready.

CYU3P_GPIF_ERR_ADDR_READ_ERR Read from DMA address thread which is not ready.

CYU3P_GPIF_ERR_ADDR_WRITE_ERR Write to DMA address thread which is not ready.

CYU3P_GPIF_ERR_INVALID_STATE GPIF state machine has reached an invalid state.

5.25.4.2 enum CyU3PPibErrorType

Enumeration of P-port error types.

Description

The P-port interface block (PIB) of the FX3 device can encounter various errors during data transfers performed across the GPIF interface. This enumerated type lists the PIB level error conditions that are notified to the user application through the CYU3P_PIB_INTR_ERROR interrupt callback. The cbArg parameter passed to the callback will indicate the PIB error code as well as the GPIF error code.

The CYU3P_GET_PIB_ERROR_TYPE macro can be used to decode the PIB error type from the callback argument.

See Also

CyU3PPibIntrType
CyU3PPibRegisterCallback
CyU3PGpifErrorType
CYU3P_GET_PIB_ERROR_TYPE

Enumerator

CYU3P_PIB_ERR_NONE No errors detected.

CYU3P_PIB_ERR_THR0_DIRECTION Bad transfer direction (read on a write socket or vice versa) error on one of the Thread 0 sockets.

CYU3P_PIB_ERR_THR1_DIRECTION Bas transfer direction (read on a write socket or vice versa) error on one of the Thread 1 sockets.

CYU3P_PIB_ERR_THR2_DIRECTION Bad transfer direction (read on a write socket or vice versa) error on one of the Thread 2 sockets.

CYU3P_PIB_ERR_THR3_DIRECTION Bas transfer direction (read on a write socket or vice versa) error on one of the Thread 3 sockets.

CYU3P_PIB_ERR_THR0_WR_OVERRUN Write overrun (write beyond available buffer size) error on one of the Thread 0 sockets.

CYU3P_PIB_ERR_THR1_WR_OVERRUN Write overrun (write beyond available buffer size) error on one of the Thread 1 sockets.

CYU3P_PIB_ERR_THR2_WR_OVERRUN Write overrun (write beyond available buffer size) error on one of the Thread 2 sockets.

CYU3P_PIB_ERR_THR3_WR_OVERRUN Write overrun (write beyond available buffer size) error on one of the Thread 3 sockets.

CYU3P_PIB_ERR_THR0_RD_UNDERRUN Read underrun (read beyond available data size) error on one of the Thread 0 sockets.

CYU3P_PIB_ERR_THR1_RD_UNDERRUN Read underrun (read beyond available data size) error on one of the Thread 1 sockets.

CYU3P_PIB_ERR_THR2_RD_UNDERRUN Read underrun (read beyond available data size) error on one of the Thread 2 sockets.

CYU3P_PIB_ERR_THR3_RD_UNDERRUN Read underrun (read beyond available data size) error on one of the Thread 3 sockets.

CYU3P_PIB_ERR_THR0_SCK_INACTIVE One of the Thread 0 sockets became inactive during transfer.

CYU3P_PIB_ERR_THR0_ADAP_OVERRUN DMA controller overrun on a write to one of the Thread 0 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

CYU3P_PIB_ERR_THR0_ADAP_UNDERRUN DMA controller underrun on a read from one of the Thread 0 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

CYU3P_PIB_ERR_THR0_RD_FORCE_END A DMA read operation from Thread 0 socket was forcibly ended by wrapping up the socket.

CYU3P_PIB_ERR_THR0_RD_BURST A socket switch was forced in the middle of a read burst from a Thread 0 socket.

CYU3P_PIB_ERR_THR1_SCK_INACTIVE One of the Thread 1 sockets became inactive during transfer.

CYU3P_PIB_ERR_THR1_ADAP_OVERRUN DMA controller overrun on a write to one of the Thread 1 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

CYU3P_PIB_ERR_THR1_ADAP_UNDERRUN DMA controller underrun on a read from one of the Thread 1 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

CYU3P_PIB_ERR_THR1_RD_FORCE_END A DMA read operation from Thread 1 socket was forcibly ended by wrapping up the socket.

CYU3P_PIB_ERR_THR1_RD_BURST A socket switch was forced in the middle of a read burst from a Thread 1 socket.

CYU3P_PIB_ERR_THR2_SCK_INACTIVE One of the Thread 2 sockets became inactive during transfer.

CYU3P_PIB_ERR_THR2_ADAP_OVERRUN DMA controller overrun on a write to one of the Thread 2 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

CYU3P_PIB_ERR_THR2_ADAP_UNDERRUN DMA controller underrun on a read from one of the Thread 2 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

CYU3P_PIB_ERR_THR2_RD_FORCE_END A DMA read operation from Thread 2 socket was forcibly ended by wrapping up the socket.

CYU3P_PIB_ERR_THR2_RD_BURST A socket switch was forced in the middle of a read burst from a Thread 2 socket.

CYU3P_PIB_ERR_THR3_SCK_INACTIVE One of the Thread 3 sockets became inactive during transfer.

CYU3P_PIB_ERR_THR3_ADAP_OVERRUN DMA controller overrun on a write to one of the Thread 3 sockets. This typically happens if the DMA controller cannot keep up with the incoming data rate.

CYU3P_PIB_ERR_THR3_ADAP_UNDERRUN DMA controller underrun on a read from one of the Thread 3 sockets. This is also a result of the DMA controller not being able to keep up with the desired interface data rate.

CYU3P_PIB_ERR_THR3_RD_FORCE_END A DMA read operation from Thread 3 socket was forcibly ended by wrapping up the socket.

CYU3P_PIB_ERR_THR3_RD_BURST A socket switch was forced in the middle of a read burst from a Thread 3 socket.

5.25.4.3 enum CyU3PPibIntrType

Enumeration of P-port interrupt event types.

Description

The P-port interface block (PIB) of the FX3 device has some interrupt sources that are unconnected with the GPIF hardware or state machine. These interrupts indicate events such as error conditions, mailbox message reception and Interface Config register updates. This enumerated type lists the various interrupt events that are valid for the PIB interrupt.

See Also

CyU3PPibIntrCb_t
CyU3PPibRegisterCallback

Enumerator

CYU3P_PIB_INTR_DLL_UPDATE Indicates that a PIB DLL lock or unlock event has occured. The cbArg parameter indicates whether the DLL is currently locked (non-zero) or unlocked (zero).

CYU3P_PIB_INTR_PPCONFIG Indicates that the PIB config register has been written to through the GPIF interface. The value written into the PP_CONFIG register is passed as the cbArg parameter. This value typically has the following format:

Bit 15 : User defined.

Bit 14 : DRQ polarity. 0 = active low, 1 = active high.

Bit 13 : User defined.

Bit 12 : DRQ override. 0 = Normal. 1 = Force DRQ on.

Bit 11 : INT polarity. 0 = active low. 1 = active high.

Bit 10 : INT value. Specifies INT state when override is on.

Bit 9 : INT override. 0 = Normal. 1 = Overridden.

Bits 8 - 7 : User defined.

Bit 7 : User defined.

Bit 6 : CFGMODE. Must be 1.

Bits 5 - 4 : User defined.

Bits 3 - 0 : User defined. Usually set to log2(burst size).

CYU3P_PIB_INTR_ERROR Indicates that an error condition has been encountered by the PIB/GPIF block. The type of error encountered is passed through the cbArg parameter. See CyU3PPibErrorType for the possible error types.

5.25.4.4 enum CyU3PPmmcEventType

List of MMC-Slave Event notifications.

Description

This type lists the various event notifications that are provided by the FX3 device when configured in the MMC-Slave mode.

See Also

CyU3PPmmcIntrCb_t

Enumerator

CYU3P_PMMC_GOIDLE_CMD GO_IDLE_STATE (CMD0) command has been received.

CYU3P_PMMC_CMD5_SLEEP CMD5(SLEEP) command has been used to place FX3 into suspend mode.

CYU3P_PMMC_CMD5_AWAKE CMD5(AWAKE) command has been used to wake FX3 from suspend mode.

CYU3P_PMMC_CMD6_SWITCH SWITCH (CMD6) command has been received.

CYU3P_PMMC_CMD12_STOP STOP_TRANSMISSION (CMD12) command has been received.

CYU3P_PMMC_CMD15_INACTIVE GO_INACTIVE_STATE (CMD15) command has been received.

CYU3P_PMMC_DIRECT_WRITE Write command has been received on the designated Direct Write socket.

CYU3P_PMMC_DIRECT_READ Read command has been received on the designated Direct Read socket.

CYU3P_PMMC_SOCKET_NOT_READY Socket being read/written by the host is not ready.

CYU3P_PMMC_CMD7_SELECT SELECT_CARD (CMD7) command has been received.

5.25.4.5 enum CyU3PPmmcState

List of MMC-Slave states.

Description

When the FX3 is configured as an MMC-Slave device, the MMC interface goes through a set of states (as defined in the MMC specification) in response to commands issued by the MMC host. This type lists the various MMC Slave states for the FX3 device.

Enumerator

CY_U3P_PMMC_IDLE Idle (reset) state.

CY_U3P_PMMC_READY Ready state.

CY_U3P_PMMC_IDENTIFICATION Identification state.

CY_U3P_PMMC_STANDBY Standby state.

CY_U3P_PMMC_TRANS Transfer (tran) state.

CY_U3P_PMMC_SENDDATA Send data (read) state.

CY_U3P_PMMC_RECVDATA Receive data (write) state.

CY_U3P_PMMC_PGM Program state.

CY_U3P_PMMC_DISCONNECT Disconnect state.

CY_U3P_PMMC_BUSTEST Bus-test state.

CY_U3P_PMMC_SLEEP Sleep state.

CY_U3P_PMMC_WAITIRQ Wait IRQ state.

CY_U3P_PMMC_INACTIVE Inactive state.

CY_U3P_PMMC_BOOT Boot state (MMC 4.3)

CY_U3P_PMMC_PREIDLE Pre-Idle state (MMC 4.4)

CY_U3P_PMMC_PREBOOT Pre-boot state (MMC 4.4)

5.25.5 Function Documentation

5.25.5.1 CyU3PReturnStatus_t CyU3PPibDeInit (void)

De-initialize the P-port interface block.

Description

This function disables and powers off the P-port interface block.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_STARTED - If the block has not been initialized.

See Also

CyU3PPibInit

5.25.5.2 CyU3PReturnStatus_t CyU3PPibInit (CyBool_t *doInit*, CyU3PPibClock_t * *pibClock*)

Initialize the P-port interface block.

Description

This function powers up the P-port interface block. This needs to be the first P-port related function call and should be called before any GPIF related calls are made.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_NOT_SUPPORTED - If the FX3 part in use does not support the PIB port.

CY_U3P_ERROR_BAD_ARGUMENT - If an incorrect/invalid clock configuration is passed.

CY_U3P_ERROR_NULL_POINTER - If the argument pibClock is a NULL.

See Also

CyU3PPibSelectMmcSlaveMode
CyU3PPibDeInit
CyU3PPibClock_t

Parameters

doInit	Whether to initialize the PIB block. Should generally be set to CyTrue.
pibClock	PIB clock configuration.

5.25.5.3 void CyU3PPibRegisterCallback (CyU3PPibIntrCb_t *cbFunc*, uint32_t *intMask*)

Register a callback function for notification of PIB interrupts.

Description

This function registers a callback function that will be called for notification of PIB interrupts and also selects the PIB interrupt sources of interest.

Return value

None

See Also

CyU3PPibIntrCb_t
CyU3PPibIntrType

Parameters

cbFunc	Callback function pointer being registered.
intMask	Bitmask representing the various interrupts callbacks to be enabled. This value is derived by ORing the various callback types of interest from the CyU3PPibIntrType enumeration.

5.25.5.4 void CyU3PPibSelectIntSources (CyBool_t *pibSockEn*, CyBool_t *gpifIntEn*, CyBool_t *pibErrEn*, CyBool_t *mboxIntEn*, CyBool_t *wakeupEn*)

Select the sources that trigger an FX3 interrupt to external processor.

Description

The FX3 device is capable of generating interrupts to the external processor connected on the GPIF port for various reasons. This function is used to select the interrupt sources allowed to interrupt the external processor.

Return value

None

Parameters

pibSockEn	Whether PIB socket DMA ready status should trigger an interrupt.
gpifIntEn	Whether GPIF state machine can trigger an interrupt.
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.
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.
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.

5.25.5.5 CyU3PReturnStatus_t CyU3PPibSelectMmcSlaveMode (void)

Select the MMC Slave mode of operation for the P-port interface block.

Description

The P-port interface block (PIB) on the FX3/FX3S devices can be configured to function in the GPIF-II mode or in the MMC Slave mode. In the MMC slave mode, the device appears as a MMC 4.2 spec compliant slave device which accepts commands from a MMC host controller.

This function is used to switch the PIB block to the MMC Slave mode (from the default GPIF-II mode). As this is a static mode selection, this function is expected to be called before the PIB block is initialized.

Return Value

CY_U3P_SUCCESS if the switch request is registered. CY_U3P_ERROR_ALREADY_STARTED if the PIB block is already active.

See Also

CyU3PPibInit
CyU3PPibDeInit

5.25.5.6 CyU3PReturnStatus_t CyU3PPibSetInterruptPriority (CyBool_t *isHigh*)

Set the priority level for PIB/GPIF interrupts in the FX3 system.

Description

By default, PIB and GPIF interrupts in the FX3 system have low priority and can be pre-empted by other interrupts such as the USB interrupt. This API is provided to switch to a higher priority non pre-emptable version of the interrupt handler; or back to the default setting.

Note

It is not advisable to set the PIB interrupt as high priority if the CYU3P_PIB_INTR_ERROR callback type is enabled.

Return value

CY_U3P_SUCCESS if the interrupt priority was updated as specified.

CY_U3P_ERROR_INVALID_CALLER if this function is called from interrupt context.

Parameters

isHigh	Whether the PIB (GPIF) interrupt is to be assigned high priority.

5.25.5.7 CyU3PReturnStatus_t CyU3PPmmcEnableDirectAccess (CyBool_t *enable*, uint8_t *wrSock*, uint8_t *rdSock*)

Enable MMC direct read/write functionality.

Description

In the normal mode of operation, the MMC-Slave interface requires a high level protocol layer that maps specific PIB sockets to corresponding data paths. Pass-through data transfers from/to SD/MMC slave devices connected on the Storage port can be facilitated through a pair of designated direct read and write sockets.

If direct read/write is enabled, the firmware is notified whenever a read/write addressed to a specific address range is received. The firmware can use the address to initiate data transfers on the Storage port, and then complete the data transfer through the direct access sockets.

Return Value

CY_U3P_SUCCESS if the direct access enable/disable is successful. CY_U3P_ERROR_NOT_STARTED if the PIB block is not enabled. CY_U3P_ERROR_INVALID_SEQUENCE if the PIB is not configured in PMMC mode. CY_U3P_ERROR_BAD_ARGUMENT if the sockets specified are not valid.

Parameters

enable	Whether to enable or disable the direct data access.
wrSock	PIB socket to be used for direct write operations (should be in the range 16 - 31).
rdSock	PIB socket to be used for direct read operations (should be in the range 0 - 15).

5.25.5.8 void CyU3PPmmcRegisterCallback (CyU3PPmmcIntrCb_t *cbFunc*)

Register a callback for notification of MMC-Slave (PMMC) events.

Description

This API registers a callback function that will be used to provide notifications of MMC-Slace related events. Please note that this callback will be invoked from the interrupt context, and cannot be used to perform any blocking operations.

Return Value

None

See Also

CyU3PPmmcIntrCb_t
CyU3PPmmcEventType

Parameters

cbFunc	Callback function pointer.

5.25.5.9 CyU3PReturnStatus_t CyU3PSetPportDriveStrength (CyU3PDriveStrengthState_t *pportDriveStrength*)

Set the IO drive strength for the Pport.

Description

The function sets the IO Drive strength for the P-port signals. The default IO drive strength is set to CY_U3P_DS_THREE_QUARTER_STRENGTH.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT - If the Drive strength requested is invalid.

See Also

CyU3PDriveStrengthState_t

Parameters

pportDrive-Strength	Desired drive strength

5.26 firmware/u3p_firmware/inc/cyu3sib.h File Reference

This file defines the data types and APIs that support SD/MMC/SDIO peripheral access on the EZ-USB FX3S device.

```
#include <cyu3os.h>
#include <cyu3dma.h>
#include <cyu3types.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PSibIntfParams
  Storage interface control parameters.
- struct CyU3PSibLunInfo
  Logical unit properties.
- struct CyU3PSibDevInfo
  Storage (SD/MMC) device properties.
- struct CyU3PSdioCardRegs
  SDIO Card Generic Information and CCCR registers.

Macros

- #define CY_U3P_SIB0_DETECT_GPIO (44)
  GPIO used for card detection on Storage port 0.
- #define CY_U3P_SIB0_WRPROT_GPIO (43)
  GPIO used for write protection detection on Storage port 0.
- #define CY_U3P_SIB1_WRPROT_GPIO (52)
  GPIO used for write protection detection on Storage port 1.

Typedefs

- typedef enum CyU3PSibPortId CyU3PSibPortId
  List of storage ports.
- typedef enum CyU3PSibCardDetect CyU3PSibCardDetect
  List of card detection methods.
- typedef enum CyU3PSibDevType CyU3PSibDevType
  List the Storage Device types supported by FX3S.
- typedef enum CyU3PSibIntfVoltage CyU3PSibIntfVoltage
  List of SD/MMC operating voltage ranges.
- typedef enum CyU3PSibEventType CyU3PSibEventType
  List of storage related events.
- typedef enum CyU3PSibLunType CyU3PSibLunType
  List of logical unit (LUN) types.
- typedef enum CyU3PSibLunLocation CyU3PSibLunLocation

List of logical unit locations.

- typedef enum CyU3PSibOpFreq CyU3PSibOpFreq

  List of storage operating frequencies.

- typedef struct CyU3PSibIntfParams CyU3PSibIntfParams_t

  Storage interface control parameters.

- typedef struct CyU3PSibLunInfo CyU3PSibLunInfo_t

  Logical unit properties.

- typedef struct CyU3PSibDevInfo CyU3PSibDevInfo_t

  Storage (SD/MMC) device properties.

- typedef void(∗ CyU3PSibEvtCbk_t)(uint8_t portId, CyU3PSibEventType evt, CyU3PReturnStatus_t status)

  Storage event callback function type.

- typedef struct CyU3PSdioCardRegs CyU3PSdioCardRegs_t

  SDIO Card Generic Information and CCCR registers.

- typedef enum CyU3PSibEraseMode CyU3PSibEraseMode

  Erase command mode.

Enumerations

- enum CyU3PSibPortId { CY_U3P_SIB_PORT_0 = 0, CY_U3P_SIB_PORT_1, CY_U3P_SIB_NUM_PORTS }

  List of storage ports.

- enum CyU3PSibCardDetect { CY_U3P_SIB_DETECT_NONE = 0, CY_U3P_SIB_DETECT_GPIO, CY_U3P_SIB_DETECT_DAT_3 }

  List of card detection methods.

- enum CyU3PSibDevType { CY_U3P_SIB_DEV_NONE = 0, CY_U3P_SIB_DEV_MMC, CY_U3P_SIB_DEV_SD, CY_U3P_SIB_DEV_SDIO, CY_U3P_SIB_DEV_SDIO_COMBO }

  List the Storage Device types supported by FX3S.

- enum CyU3PSibIntfVoltage { CY_U3P_SIB_VOLTAGE_NORMAL = 0, CY_U3P_SIB_VOLTAGE_LOW }

  List of SD/MMC operating voltage ranges.

- enum CyU3PSibEventType { CY_U3P_SIB_EVENT_INSERT = 0, CY_U3P_SIB_EVENT_REMOVE, CY_U3P_SIB_EVENT_XFER_CPLT, CY_U3P_SIB_EVENT_SDIO_INTR, CY_U3P_SIB_EVENT_RELEASE, CY_U3P_SIB_EVENT_DATA_ERROR, CY_U3P_SIB_EVENT_ABORT }

  List of storage related events.

- enum CyU3PSibLunType { CY_U3P_SIB_LUN_RSVD = 0xAB, CY_U3P_SIB_LUN_BOOT = 0xBB, CY_U3P_SIB_LUN_DATA = 0xDA }

  List of logical unit (LUN) types.

- enum CyU3PSibLunLocation { CY_U3P_SIB_LOCATION_USER = 0, CY_U3P_SIB_LOCATION_BOOT1, CY_U3P_SIB_LOCATION_BOOT2 }

  List of logical unit locations.

- enum CyU3PSibOpFreq { CY_U3P_SIB_FREQ_400KHZ, CY_U3P_SIB_FREQ_20MHZ, CY_U3P_SIB_FREQ_26MHZ, CY_U3P_SIB_FREQ_52MHZ, CY_U3P_SIB_FREQ_104MHZ }

  List of storage operating frequencies.

- enum CyU3PSibEraseMode { CY_U3P_SIB_ERASE_STANDARD = 0, CY_U3P_SIB_ERASE_SECURE, CY_U3P_SIB_ERASE_TRIM_STEP1, CY_U3P_SIB_ERASE_TRIM_STEP2 }

  Erase command mode.

Functions

- CyU3PReturnStatus_t CyU3PSibSetIntfParams (uint8_t portId, CyU3PSibIntfParams_t *intfParams)

  Define the interface parameters for a storage port.
- CyU3PReturnStatus_t CyU3PSibStart (void)

  Start the storage driver and try to initialize any storage devices connected.
- void CyU3PSibStop (void)

  Stop the storage driver on the FX3S device.
- CyU3PReturnStatus_t CyU3PSibInit (uint8_t portId)

  Initialize the SD/MMC/SDIO device on the specified port.
- void CyU3PSibDeInit (uint8_t portId)

  De-Initialize the storage device on the specified port.
- CyU3PReturnStatus_t CyU3PSibRegisterCbk (CyU3PSibEvtCbk_t sibEvtCbk)

  Register a function for notification of storage related events.
- CyU3PReturnStatus_t CyU3PSibQueryDevice (uint8_t portId, CyU3PSibDevInfo_t *devInfo_p)

  Query storage device information.
- CyU3PReturnStatus_t CyU3PSibQueryUnit (uint8_t portId, uint8_t unitId, CyU3PSibLunInfo_t *unitInfo_p)

  Query storage partition information.
- CyU3PReturnStatus_t CyU3PSibGetCSD (uint8_t portId, uint8_t *csd_buffer)

  Get the CSD register content for an SD/MMC card connected.
- CyU3PReturnStatus_t CyU3PSibGetMMCExtCsd (uint8_t portId, uint8_t *buffer_p)

  Read the Extended CSD register from an MMC card.
- CyU3PReturnStatus_t CyU3PSibSendSwitchCommand (uint8_t portId, uint32_t argument, uint32_t *resp_p)

  Send a SWITCH command to update the Ext. CSD register on an MMC device.
- void CyU3PSibSetBlockLen (uint8_t portId, uint16_t blkLen)

  Define the block length to be used for storage data transfers.
- CyU3PReturnStatus_t CyU3PSibReadWriteRequest (CyBool_t isRead, uint8_t portId, uint8_t unitId, uint16_t numBlks, uint32_t blkAddr, uint8_t socket)

  Initiates a read/write data request.
- CyU3PReturnStatus_t CyU3PSibCommitReadWrite (uint8_t portId)

  Commits any pending read/write transaction.
- CyU3PReturnStatus_t CyU3PSibSetWriteCommitSize (uint8_t portId, uint32_t numSectors)

  Set the sector granularity at which write operations are committed to the SD/MMC storage.
- CyU3PReturnStatus_t CyU3PSibAbortRequest (uint8_t portId)

  Abort an ongoing transaction.
- CyU3PReturnStatus_t CyU3PSibEraseBlocks (uint8_t portId, uint16_t numEraseUnits, uint32_t startAddr, CyU3PSibEraseMode mode)

  Erase blocks on the storage card.
- CyU3PReturnStatus_t CyU3PSibGetCardStatus (uint8_t portId, uint32_t *status_p)

  Function to get current status of the SD/MMC card.
- CyU3PReturnStatus_t CyU3PSibPartitionStorage (uint8_t portId, uint8_t partCount, uint32_t *partSizes_p, uint8_t *partType_p)

  Partition a storage device into multiple logical units.
- CyU3PReturnStatus_t CyU3PSibRemovePartitions (uint8_t portId)

  Remove all partitions and re-unify a storage device.
- CyU3PReturnStatus_t CyU3PSibLockUnlockCard (uint8_t portId, CyBool_t lockCard, CyBool_t clrPasswd, uint8_t *passwd, uint8_t passwdLen)

  Lock/Unlock an SD Card.
- CyU3PReturnStatus_t CyU3PSibSetPasswd (uint8_t portId, CyBool_t lockCard, uint8_t *passwd, uint8_t passwdLen, uint8_t *newPasswd, uint8_t newPasswdLen)

  Set/replace password on an SD Card.

- CyU3PReturnStatus_t CyU3PSibRemovePasswd (uint8_t portId, uint8_t *passwd, uint8_t passwdLen)

  Clear the password on an SD Card.
- CyU3PReturnStatus_t CyU3PSibForceErase (uint8_t portId)

  Force erase the user content and the lock/unlock related information on an SD Card.
- CyU3PReturnStatus_t CyU3PSibVendorAccess (uint8_t portId, uint8_t cmd, uint32_t cmdArg, uint8_t respLen, uint8_t *respData, uint16_t numBlks, CyBool_t isRead, uint8_t socket)

  Vendor SD/MMC command access.
- CyU3PReturnStatus_t CyU3PSdioCardReset (uint8_t portId)

  Reset an SDIO card.
- CyU3PReturnStatus_t CyU3PSdioQueryCard (uint8_t portId, CyU3PSdioCardRegs_t *data)

  Fetch SDIO card information and Card common register information.
- CyU3PReturnStatus_t CyU3PSdioByteReadWrite (uint8_t portId, uint8_t functionNumber, uint8_t isWrite, uint8_t readAfterWriteFlag, uint32_t registerAddr, uint8_t *data)

  Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.
- CyU3PReturnStatus_t CyU3PSdioExtendedReadWrite (uint8_t portId, uint8_t functionNumber, uint8_t isWrite, uint8_t blockMode, uint8_t opCode, uint32_t registerAddr, uint16_t transferCount, uint16_t sckId)

  Perform Extended I/O operation (Multi-Byte/Block) to/from SDIO card using CMD53.
- CyU3PReturnStatus_t CyU3PSdioSuspendResumeFunction (uint8_t portId, uint8_t functionNumber, uint8_t operation, uint8_t *isDataAvailable)

  Suspend or Resume SDIO I/O Function.
- CyU3PReturnStatus_t CyU3PSdioAbortFunctionIO (uint8_t portId, uint8_t functionNumber)

  Abort an ongoing extended read or extended write operation.
- CyU3PReturnStatus_t CyU3PSdioInterruptControl (uint8_t portId, uint8_t functionNumber, uint8_t operation, uint8_t *intEnReg)

  Enable or Disable Interrupts on the SDIO Card.
- CyU3PReturnStatus_t CyU3PSdioGetCISAddress (uint8_t portId, uint8_t functionNumber, uint32_t *addressCIS)

  Get Address for the CIS for the specified card function.
- CyU3PReturnStatus_t CyU3PSdioGetTuples (uint8_t portId, uint8_t funcNo, uint8_t tupleId, uint32_t addrCIS, uint8_t *buffer, uint8_t *tupleSize)

  Read data for a CIS Tuple into a buffer.
- CyU3PReturnStatus_t CyU3PSdioSetBlockSize (uint8_t portId, uint8_t funcNo, uint16_t blockSize)

  Set the block size for an IO function.
- CyU3PReturnStatus_t CyU3PSdioGetBlockSize (uint8_t portId, uint8_t funcNo, uint16_t *blockSize)

  Get block size which has been set for IO function to the device.
- CyU3PReturnStatus_t CyU3PSdioReadWaitEnable (uint8_t portId, uint8_t isRdWtEnable)

  Enable Read Wait on the SDIO BUS.

5.26.1 Detailed Description

This file defines the data types and APIs that support SD/MMC/SDIO peripheral access on the EZ-USB FX3S device. **Description**

The EZ-USB FX3S device is an extension to the FX3 device, which has the capability to talk to SD/MMC/SDIO peripherals.

The storage module in FX3S firmware manages accesses to SD, MMC and SDIO devices. The storage module is composed of a storage driver and convenience API; and is compiled into a separate static library. The storage library (cyu3sport.a) only needs to be linked in by FX3S applications that make use of the SD/MMC/SDIO interfaces.

Note

The storage driver internally makes use of GPIO functions. Therefore, applications linking with the storage library will also need to link with the serial peripheral library (cyu3lpp.a).

5.26.2 Typedef Documentation

5.26.2.1 typedef struct CyU3PSdioCardRegs CyU3PSdioCardRegs_t

SDIO Card Generic Information and CCCR registers.

Description

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

5.26.2.2 typedef enum CyU3PSibCardDetect CyU3PSibCardDetect

List of card detection methods.

Description

The card insertion and removal detection can be based on either GPIO card detect or based on signal changes on the DAT3 line.

5.26.2.3 typedef struct CyU3PSibDevInfo CyU3PSibDevInfo_t

Storage (SD/MMC) device properties.

Description

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

See Also

CyU3PSibDevType

5.26.2.4 typedef enum CyU3PSibDevType CyU3PSibDevType

List the Storage Device types supported by FX3S.

Description

This enumeration lists the various storage device types supported by the storage module in FX3S firmware.

5.26.2.5 typedef enum CyU3PSibEraseMode CyU3PSibEraseMode

Erase command mode.

Description

This enumeration lists the possible modes for the erase command. These modes are only applicable to eMMC media, and only the standard erase mode is supported for SD cards.

5.26.2.6 typedef enum CyU3PSibEventType CyU3PSibEventType

List of storage related events.

Description

The storage driver propagates events of interest such as card insertion/removal and data transfer completion through a callback function.

5.26.2.7 typedef void(∗ CyU3PSibEvtCbk_t)(uint8_t portId,CyU3PSibEventType evt,CyU3PReturnStatus_t status)

Storage event callback function type.

Description

This function type defines a callback for the storage events as well as for the status of the storage read/write operations.

See Also

CyU3PSibEventType
CyU3PSibRegisterCbk

5.26.2.8 typedef struct CyU3PSibIntfParams CyU3PSibIntfParams_t

Storage interface control parameters.

Description

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

5.26.2.9 typedef enum CyU3PSibIntfVoltage CyU3PSibIntfVoltage

List of SD/MMC operating voltage ranges.

Description

The storage host driver supports storage functionality at both the normal voltage (3.3V) and low voltage (1.8V). Low voltage is necessary for the use of the SDR50 signaling rate on the SD 3.0 interface.

5.26.2.10 typedef struct CyU3PSibLunInfo CyU3PSibLunInfo_t

Logical unit properties.

Description

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

See Also

CyU3PSibLunType
CyU3PSibLunLocation

5.26.2.11 typedef enum CyU3PSibLunLocation CyU3PSibLunLocation

List of logical unit locations.

Description

MMC devices can support one or more boot partitions in addition to the user data area, and a given storage LUN may be located in the boot partition or in the user data area. This enumerated type lists the possible location of a logical unit.

5.26.2.12 typedef enum CyU3PSibLunType CyU3PSibLunType

List of logical unit (LUN) types.

Description

The storage devices can be partitioned to store boot images as well as for storing user content. The boot partitions are only accessible under boot or boot update modes.

5.26.2.13 typedef enum CyU3PSibOpFreq CyU3PSibOpFreq

List of storage operating frequencies.

Description

This enumerated type lists the various operating frequencies supported on the storage port by the FX3S device. This type is used to set a constraint on the maximum operating frequency for the storage devices.

5.26.2.14 typedef enum CyU3PSibPortId CyU3PSibPortId

List of storage ports.

Description

The FX3S device has two storage ports which support SD/MMC/SDIO cards to be attached. These storage ports can be accessed independently of each other.

5.26.3 Enumeration Type Documentation

5.26.3.1 enum CyU3PSibCardDetect

List of card detection methods.

Description

The card insertion and removal detection can be based on either GPIO card detect or based on signal changes on the DAT3 line.

Enumerator

CY_U3P_SIB_DETECT_NONE Don't use any card detection mechanism

CY_U3P_SIB_DETECT_GPIO Use a GPIO connected to a micro-switch on the socket. This is only supported for the S0 storage port.

CY_U3P_SIB_DETECT_DAT_3 Use voltage changes on the DAT[3] pin for card detection.

5.26.3.2 enum CyU3PSibDevType

List the Storage Device types supported by FX3S.

Description

This enumeration lists the various storage device types supported by the storage module in FX3S firmware.

Enumerator

CY_U3P_SIB_DEV_NONE No device

CY_U3P_SIB_DEV_MMC MMC/eMMC device

CY_U3P_SIB_DEV_SD SD Memory card

CY_U3P_SIB_DEV_SDIO SDIO IO only Device.

CY_U3P_SIB_DEV_SDIO_COMBO SDIO Combo Device. Data transfers to the combo card are not supported.

5.26.3.3 enum CyU3PSibEraseMode

Erase command mode.

Description

This enumeration lists the possible modes for the erase command. These modes are only applicable to eMMC media, and only the standard erase mode is supported for SD cards.

Enumerator

CY_U3P_SIB_ERASE_STANDARD Standard erase mode.

CY_U3P_SIB_ERASE_SECURE Secure erase mode. Only supported for eMMC.

CY_U3P_SIB_ERASE_TRIM_STEP1 Secure trim STEP1. Mark the blocks for secure erase.

CY_U3P_SIB_ERASE_TRIM_STEP2 Secure trim STEP2. Erase blocks marked in trim STEP1.

5.26.3.4 enum CyU3PSibEventType

List of storage related events.

Description

The storage driver propagates events of interest such as card insertion/removal and data transfer completion through a callback function.

Enumerator

CY_U3P_SIB_EVENT_INSERT Card Insert Event

CY_U3P_SIB_EVENT_REMOVE Card Remove Event

CY_U3P_SIB_EVENT_XFER_CPLT Data transfer completion. The status field will indicate success/failure.

CY_U3P_SIB_EVENT_SDIO_INTR SDIO interrupt event.

CY_U3P_SIB_EVENT_RELEASE Notification that device is free for access: Not supported.

CY_U3P_SIB_EVENT_DATA_ERROR Errors like CRC16, Data Timeout and Data Error handler event.

CY_U3P_SIB_EVENT_ABORT Transaction aborted event.

5.26.3.5 enum CyU3PSibIntfVoltage

List of SD/MMC operating voltage ranges.

Description

The storage host driver supports storage functionality at both the normal voltage (3.3V) and low voltage (1.8V). Low voltage is necessary for the use of the SDR50 signaling rate on the SD 3.0 interface.

Enumerator

CY_U3P_SIB_VOLTAGE_NORMAL Normal voltage, 3.3 V.

CY_U3P_SIB_VOLTAGE_LOW Low voltage, 1.8 V.

5.26.3.6 enum CyU3PSibLunLocation

List of logical unit locations.

Description

MMC devices can support one or more boot partitions in addition to the user data area, and a given storage LUN may be located in the boot partition or in the user data area. This enumerated type lists the possible location of a logical unit.

Enumerator

CY_U3P_SIB_LOCATION_USER The LUN is located in the user data area.

CY_U3P_SIB_LOCATION_BOOT1 The LUN is located in the first boot partition.

CY_U3P_SIB_LOCATION_BOOT2 The LUN is located in the second boot partition.

5.26.3.7 enum CyU3PSibLunType

List of logical unit (LUN) types.

Description

The storage devices can be partitioned to store boot images as well as for storing user content. The boot partitions are only accessible under boot or boot update modes.

Enumerator

CY_U3P_SIB_LUN_RSVD LUN type reserved for future use.

CY_U3P_SIB_LUN_BOOT LUN holding FX3S Boot code

CY_U3P_SIB_LUN_DATA LUN holding user data

5.26.3.8 enum CyU3PSibOpFreq

List of storage operating frequencies.

Description

This enumerated type lists the various operating frequencies supported on the storage port by the FX3S device. This type is used to set a constraint on the maximum operating frequency for the storage devices.

Enumerator

CY_U3P_SIB_FREQ_400KHZ 400 KHz. Used for device initialization only. Cannot be set as the maximum frequency.

CY_U3P_SIB_FREQ_20MHZ 20 MHz SDR.

CY_U3P_SIB_FREQ_26MHZ 26 MHz SDR.

CY_U3P_SIB_FREQ_52MHZ 52 MHz SDR.

CY_U3P_SIB_FREQ_104MHZ Max. frequency possible: 104 MHz SDR or 52 MHz DDR.

5.26.3.9 enum CyU3PSibPortId

List of storage ports.

Description

The FX3S device has two storage ports which support SD/MMC/SDIO cards to be attached. These storage ports can be accessed independently of each other.

Enumerator

CY_U3P_SIB_PORT_0 Storage Port 0

CY_U3P_SIB_PORT_1 Storage Port 1

CY_U3P_SIB_NUM_PORTS Maximum number of storage ports supported

5.26.4 Function Documentation

5.26.4.1 CyU3PReturnStatus_t CyU3PSdioAbortFunctionIO (uint8_t *portId,* uint8_t *functionNumber*)

Abort an ongoing extended read or extended write operation.

Description

This function is used abort an ongoing extended I/O operation.

Return value

CY_U3P_SUCCESS

CY_U3P_ERROR_BAD_ARGUMENT

Parameters

portId	Port ID for card on which I/O abort is to be executed.
functionNumber	SDIO Card Function number on which I/O is to be aborted.

5.26.4.2 CyU3PReturnStatus_t CyU3PSdioByteReadWrite (uint8_t *portId,* uint8_t *functionNumber,* uint8_t *isWrite,* uint8_t *readAfterWriteFlag,* uint32_t *registerAddr,* uint8_t * *data*)

Perform a Direct I/O operation (Single Byte) to an SDIO card using CMD52.

Description

This function is used to read or write a single byte of data from/to a register on the SDIO card. When the readAfter-Write flag is set, the data from the register is read back after the write has been completed.

Return value

CY_U3P_SUCCESS if the I/O operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the I/O operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support the operation.

CY_U3P_ERROR_BAD_ARGUMENT if function number or register address is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_BAD_CMD_ARG if the command argument was out of the allowed range for the card.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port ID for card on which CMD52 operation is to be executed.
functionNumber	SDIO Card Function number (0-7) for the Byte IO operation.
isWrite	CY_U3P_SDIO_WRITE indicates Write a operation. CY_U3P_SDIO_READ indicates a Read operation.
readAfterWrite-Flag	CY_U3P_SDIO_READ_AFTER_WRITE indicates request for Read back of register after Write.

registerAddr	Register address to which IO operation is to be performed.
data	In case of Write operation, the byte of data to be written to the card. In case of Read (or Write with readAfterWriteFlag=1) the data read from the SDIO card will be returned through the same parameter.

5.26.4.3 CyU3PReturnStatus_t CyU3PSdioCardReset (uint8_t *portId*)

Reset an SDIO card.

Description

This function resets an I/O only SDIO card (or the I/O portion of a combo card) by writing 1 to the RES bit in the CCCR. This function does not reset the memory portion in case of a combo card.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device on the addressed port is not an SDIO device.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive sates.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port ID for card to be Reset.

5.26.4.4 CyU3PReturnStatus_t CyU3PSdioExtendedReadWrite (uint8_t *portId,* uint8_t *functionNumber,* uint8_t *isWrite,* uint8_t *blockMode,* uint8_t *opCode,* uint32_t *registerAddr,* uint16_t *transferCount,* uint16_t *sckId*)

Perform Extended I/O operation (Multi-Byte/Block) to/from SDIO card using CMD53.

Description

This API reads or writes multiple bytes of data from/to the address specified. The length of data to be written needs to be specified as the parameter. The transfer size can be specified in the form of a byte count or a block count. The API provides the option of auto-incrementing the address after writing or reading each data byte/block. The data is read/written to/from the dma channel associated with the socket ID passed in for the operation. This operation should use one of the SIB Sockets as producer (for read) or consumer (for write).

Transferring infinite blocks of data (by setting transferCount to 0 in block mode) is not supported on the FX3S devices. If transferCount is set to 0, FX3S will transfer 512 bytes of data in Multi-Byte transfer mode, but will throw an error in Multi-Block mode.

The following table shows the number of bytes and blocks transferred based on transferCount values:

transferCount Value	0x000	0x001	0x002	...	0x1FF
Bytes Transfered	512	1	2		511
Blocks Transferred	illegal	1	2		511

Return value

CY_U3P_SUCCESS if the IO operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support CMD53 operations.

CY_U3P_ERROR_BAD_ARGUMENT if the function number or register address is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_BAD_CMD_ARG if the command argument was out of the allowed range for the card.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port ID for card on which CMD53 operation is to be executed.
functionNumber	SDIO Card Function number (0-7) for the Extended IO operation.
isWrite	CY_U3P_SDIO_WRITE indicates a Write operation. CY_U3P_SDIO_READ indicates a Read operation.
blockMode	CY_U3P_SDIO_RW_BYTE_MODE indicates multi-byte transfer. CY_U3P_SDIO_RW_BLOCK_MODE indicates multi-block (1 or more) transfer.
opCode	CY_U3P_SDIO_RW_ADDR_FIXED indicates all I/O to a single address (FIFO access); CY_U3P_SDIO_RW_ADDR_AUTO_INCREMENT indicates I/O to incrementing addresses (RAM like data buffer).
registerAddr	Register address to which I/O operation is to be performed. This will be the starting address in case of auto-increment mode.
transferCount	Number of Bytes or Blocks to be transferred.
sckId	DMA Socket to be used to produce/consume data for the I/O operation.

5.26.4.5 CyU3PReturnStatus_t CyU3PSdioGetBlockSize (uint8_t *portId*, uint8_t *funcNo*, uint16_t * *blockSize*)

Get block size which has been set for IO function to the device.

Description

This function gets the block size for a function by reading the block size registers for the function. The API also updates the saved transfer block size in the driver which is used in case of extended transfers.

Return value

CY_U3P_SUCCESS if the IO operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support the operation.

CY_U3P_ERROR_BAD_ARGUMENT if the function number is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_BAD_CMD_ARG if the command's argument was out of the allowed range for the card.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port number for the card on which operation is to be executed.
funcNo	Function whose block size is to be set.

blockSize	Pointer for buffer through which the block size will be returned.

5.26.4.6 CyU3PReturnStatus_t CyU3PSdioGetCISAddress (uint8_t *portId*, uint8_t *functionNumber*, uint32_t * *addressCIS*)

Get Address for the CIS for the specified card function.

Description

This function gets the Card Information Structure base address for the specified function on the SDIO card.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs while reading the CIS address bytes.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support this operation.

CY_U3P_ERROR_BAD_ARGUMENT if the function number is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port number for the card on which operation is to be executed.
functionNumber	Function number whose CIS address is to be fetched.
addressCIS	Value of the Function's 17-Bit CIS Address

5.26.4.7 CyU3PReturnStatus_t CyU3PSdioGetTuples (uint8_t *portId*, uint8_t *funcNo*, uint8_t *tupleId*, uint32_t *addrCIS*, uint8_t * *buffer*, uint8_t * *tupleSize*)

Read data for a CIS Tuple into a buffer.

Description

This function reads out a CIS Tuple from the card's CIS area into a buffer. The size of the tuple is returned in the tupleSize parameter. The buffer memory should be initialized before being provided to this call. As per the SDIO specification, no SDIO tuple can have more than 255 bytes of data.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support this operation.

CY_U3P_ERROR_BAD_ARGUMENT if the function number is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port number for the card on which operation is to be executed.
funcNo	Function whose Tuple Data is to be fetched.
tupleId	TUPLE_CODE for which data is to be fetched.
addrCIS	Address to the Function's CIS.
buffer	Pointer for buffer into which data will be read into.
tupleSize	Size of tuple body as read from the TUPLE_LINK

5.26.4.8 CyU3PReturnStatus_t CyU3PSdioInterruptControl (uint8_t *portId*, uint8_t *functionNumber*, uint8_t *operation*, uint8_t * *intEnReg*)

Enable or Disable Interrupts on the SDIO Card.

Description

This function is used to enable or disable interrupts on the SDIO card. A request to enable interrupts for an SDIO Function (1-7) will automatically enable the card's Interrupt Master.

Return value

CY_U3P_SUCCESS if the register update is successful.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the register update.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support this operation.

CY_U3P_ERROR_BAD_ARGUMENT if the function number is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_BAD_CMD_ARG if the command argument was out of the allowed range for the card.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

See Also

CyU3PSdioCheckInterruptEnableStatus

Parameters

portId	Port number for card on which interrupt control operation is to be executed.
functionNumber	Function number (1-7) or CY_U3P_SDIO_INT_MASTER (for card interrupt master), for which interrupt control operation is requested
operation	CY_U3P_SDIO_DISABLE_INT indicates interrupt for function (or global) needs to be disabled. CY_U3P_SDIO_ENABLE_INT indicates interrupt for function (or global) needs to be enabled. CY_U3P_SDIO_CHECK_INT_ENABLE_REG indicates a request to read and return the interrupt enable register from the card's common registers.
intEnReg	Value of card's Interrupt Enable register post execution of enable / disable / check operation. Status for current function can be checked using CyU3PSdioCheckInterruptEnableStatus.

5.26.4.9 CyU3PReturnStatus_t CyU3PSdioQueryCard (uint8_t *portId*, CyU3PSdioCardRegs_t * *data*)

Fetch SDIO card information and Card common register information.

Description

This function returns an CYU3PSdioCardRegs_t object which contains card common register information from an

initialized SDIO card on the port specified by portId.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_NOT_SUPPORTED if the device on the addressed port is not an SDIO device.

See Also

CyU3PSdioCardRegs_t

Parameters

portId	Port id on which SDIO card is connected.
data	Output parameter to be filled with SDIO register information.

5.26.4.10 CyU3PReturnStatus_t CyU3PSdioReadWaitEnable (uint8_t *portId*, uint8_t *isRdWtEnable*)

Enable Read Wait on the SDIO BUS.

Description

This function triggers Read-Wait on card which supports the feature using the DAT[2] of the SDIO Data bus.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support the operation.

Parameters

portId	Port number for the card on which operation is to be executed.
isRdWtEnable	1 to Trigger Read-Wait on the Data Bus, 0 to restart I/O on the Data Bus

5.26.4.11 CyU3PReturnStatus_t CyU3PSdioSetBlockSize (uint8_t *portId*, uint8_t *funcNo*, uint16_t *blockSize*)

Set the block size for an IO function.

Description

This function sets the block size for a function. The block size set should be lower than the maximum block size value read from the CISTPL_FUNCE CIS tuple for the function. The value set to the card using this API is saved in the driver and is used as the transfer block size in case of extended transfers.

Return value

CY_U3P_SUCCESS if the IO operation completes successfully.

CY_U3P_ERROR_TIMEOUT if a timeout occurs during the operation.

CY_U3P_ERROR_NOT_SUPPORTED if the device does not support the operation.

CY_U3P_ERROR_BAD_ARGUMENT if the function number is out of range.

CY_U3P_ERROR_CARD_NOT_ACTIVE if card has not been intialized or is in Standby or Inactive states.

CY_U3P_ERROR_BAD_CMD_ARG if the command's argument was out of the allowed range for the card.

CY_U3P_ERROR_INVALID_FUNCTION if an invalid function number was requested.

CY_U3P_ERROR_UNKNOWN if a general or unknown error occurred during the operation.

CY_U3P_ERROR_ILLEGAL_CMD if the command is not legal for the card state.

CY_U3P_ERROR_CRC if the CRC check of the previous command failed.

Parameters

portId	Port number for the card on which operation is to be executed.
funcNo	Function whose block size is to be set.
blockSize	Block size to be set for the function.

5.26.4.12 CyU3PReturnStatus_t CyU3PSdioSuspendResumeFunction (uint8_t *portId*, uint8_t *functionNumber*, uint8_t *operation*, uint8_t * *isDataAvailable*)

Suspend or Resume SDIO I/O Function.

Description

This function is used to suspend or resume the specified SDIO Function. Suspend and resume needs to be supported by the card to which these calls are being addressed. Support for suspend and resume can be ascertained by looking at the CY_U3P_SDIO_CARD_CAPABLITY_SBS bit of the card's card-capability register. When the function is resumed, it also checks if any data is available.

Note

This function is not supported in the SDK 1.3 Beta release.

Return value

CY_U3P_SUCCESS

CY_U3P_ERROR_BAD_ARGUMENT

CY_U3P_ERROR_NOT_SUPPORTED

CY_U3P_ERROR_RESUME_FAILED

Parameters

portId	Port ID for card on which suspend/resume operation is to be executed.
functionNumber	SDIO Card Function number to be suspended.
operation	CY_U3P_SDIO_SUSPEND indicates a request to Suspend the function. CY_U3P_SDIO_RESUME indicates a request to Resume the function.
isDataAvailable	Non-zero if data is available after resuming the function

5.26.4.13 CyU3PReturnStatus_t CyU3PSibAbortRequest (uint8_t *portId*)

Abort an ongoing transaction.

Description

This function is used to abort any ongoing transactions on the specified port. Prior to calling this function, the corresponding DMA channel should be reset.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

Parameters

portId	Port on which the transaction is to be aborted.

5.26.4.14 CyU3PReturnStatus_t CyU3PSibCommitReadWrite (uint8_t *portId*)

Commits any pending read/write transaction.

Description

Function to commit a pending read/write operation to the SD/MMC storage; by making use of the STOP_TRANSMISSION command.

Return value

CY_U3P_SUCCESS if the operation was committed successfully.

CY_U3P_ERROR_NOT_STARTED if the storage driver has not been started.

CY_U3P_ERROR_INVALID_DEV if there is no device connected to the port.

See Also

CyU3PSibReadWriteRequest
CyU3PSibSetWriteCommitSize

Parameters

portId	Port on which the operation is to be committed.

5.26.4.15 void CyU3PSibDeInit (uint8_t *portId*)

De-Initialize the storage device on the specified port.

Description

Request to selectively de-initialize the devices connected on one of the storage ports.

Return value

None

See Also

CyU3PSibInit
CyU3PSibStop

Parameters

portId	The storage port which has to be uninitialized.

5.26.4.16 CyU3PReturnStatus_t CyU3PSibEraseBlocks (uint8_t *portId,* uint16_t *numEraseUnits,* uint32_t *startAddr,* CyU3PSibEraseMode *mode*)

Erase blocks on the storage card.

Description

This function is used to erase the specified number of erase units on an SD card. This operation is currently supported only for SD cards, and not for MMC/eMMC devices.

Return value

CY_U3P_SUCCESS - if the operation completed successfully.

CY_U3P_ERROR_NOT_STARTED - if the storage driver has not been started.

CY_U3P_ERROR_INVALID_DEV - if no storage device is found on the specified port.

CY_U3P_ERROR_NOT_SUPPORTED - if this operation is requested on a device other than SD card.

CY_U3P_ERROR_INVALID_ADDR - if the address given is not valid.

CY_U3P_ERROR_TIMEOUT - if the operation timed out while erasing the blocks.

CY_U3P_ERROR_DEVICE_BUSY - if the device is processing another request.

See Also

CyU3PSibEraseMode

Parameters

portId	Port Id on which the erase operation is to be carried out.
numEraseUnits	Number of Erase Units to be erased.
startAddr	Starting address from which the blocks are to be erased.
mode	Type of erase operation to be performed.

5.26.4.17 CyU3PReturnStatus_t CyU3PSibForceErase (uint8_t *portId*)

Force erase the user content and the lock/unlock related information on an SD Card.

Description

This function is used force erase the user content on the SD Card and also clear the current password and unlock the card.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_CARD_FORCE_ERASE if the operation failed while force erasing.

Parameters

portId	Port on which the force erase is to be done.

5.26.4.18 CyU3PReturnStatus_t CyU3PSibGetCardStatus (uint8_t *portId,* uint32_t * *status_p*)

Function to get current status of the SD/MMC card.

Description

This function sends the CMD13 (SEND_STATUS) command to the SD/MMC device connected on the specified storage port and provides the card status response that is received. This API can be used to check whether the storage device is accessible and is in the TRAN state where it can receive new commands.

Return value

CY_U3P_SUCCESS - if a R1 response was received from the storage device.

CY_U3P_ERROR_NOT_STARTED - if the storage driver has not been started.

CY_U3P_ERROR_INVALID_DEV - if no storage device is found on the specified port.

CY_U3P_ERROR_TIMEOUT - if there is a response timeout or the card is signalling busy condition.

CY_U3P_ERROR_DEVICE_BUSY - if the device is processing another request.

Parameters

portId	Port id on which the card is connected.
status_p	Return parameter through which the card status is retrieved.

5.26.4.19 CyU3PReturnStatus_t CyU3PSibGetCSD (uint8_t *portId*, uint8_t * *csd_buffer*)

Get the CSD register content for an SD/MMC card connected.

Description

This function is used to access the CSD register data of a card on a particular port. This function only returns the CSD value that is read during device initialization.

Return value

CY_U3P_SUCCESS if the query function is successful.

CY_U3P_ERROR_NOT_STARTED if the storage driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT if the portId or csd_buffer argument is invalid.

CY_U3P_ERROR_INVALID_DEV if the storage device does not exist.

Parameters

portId	Port on which to query the CSD.
csd_buffer	Buffer to be filled with CSD content.

5.26.4.20 CyU3PReturnStatus_t CyU3PSibGetMMCExtCsd (uint8_t *portId*, uint8_t * *buffer_p*)

Read the Extended CSD register from an MMC card.

Description

This function reads the extended CSD register content from an MMC device connected to FX3S.

Return value

CY_U3P_SUCCESS if the read is successful.

CY_U3P_ERROR_BAD_ARGUMENT if the portId or buffer_p argument is invalid.

CY_U3P_ERROR_INVALID_DEV if the device addressed is not an MMC card.

CY_U3P_ERROR_DEVICE_BUSY if the storage device is performing a data transfer.

See Also

CyU3PSibSendSwitchCommand

Parameters

portId	Port on which to read the Ext. CSD register.
buffer_p	Buffer to read the register value into. Should be able to hold 512 bytes.

5.26.4.21 CyU3PReturnStatus_t CyU3PSibInit (uint8_t *portId*)

Initialize the SD/MMC/SDIO device on the specified port.

Description

This function initializes any storage device connected on the specified port. Also creates the DMA Channels and locks required for the driver to access the device safely.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_SIB_INIT if the DMA channel creation associated with the storage port fails.

See Also

CyU3PSibDeInit
CyU3PSibStart

5.26.4.22 CyU3PReturnStatus_t CyU3PSibLockUnlockCard (uint8_t *portId*, CyBool_t *lockCard*, CyBool_t *clrPasswd*, uint8_t * *passwd*, uint8_t *passwdLen*)

Lock/Unlock an SD Card.

Description

This function is used to lock or unlock an SD Card on the specified port.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_CARD_LOCK_UNLOCK if the operation to lock/unlock the card fails.

Parameters

portId	Port on which the lock/unlock operation is to be done.
lockCard	Specifies whether to lock (1) or unlock (0) the card.
clrPasswd	Used along with the Unlock card to clear the password.
passwd	The current password set in the card.
passwdLen	The current password length in bytes.

5.26.4.23 CyU3PReturnStatus_t CyU3PSibPartitionStorage (uint8_t *portId*, uint8_t *partCount*, uint32_t * *partSizes_p*, uint8_t * *partType_p*)

Partition a storage device into multiple logical units.

Description

The available storage on a SD/MMC device connected to FX3S can be partitioned into multiple logical units which can be separately accessed. This function is used to setup the partitions as required. The number of partitions to be created, the sizes for all except the last partition; and the types associated with each partition are specified as parameters. The size of the last partition is inferred as the number of remaining blocks available on the storage device, and does not need to be specified.

As a custom partitioning scheme is used by the firmware, partitions created by FX3S cannot be identified by another SD/MMC host controller. Therefore, this feature should only be used in cases where there is no need for the storage device to be read through another controller.

Note

The FX3S boot loader will only be able to boot from a storage partition at the top of the available storage. Therefore, it is not useful to set any of the partitions other than 0, as CY_U3P_SIB_LUN_BOOT partitions.

See Also

CyU3PSibLunInfo_t
CyU3PSibLunType
CyU3PSibQueryUnit
CyU3PSibRemovePartitions

Return value

CY_U3P_SUCCESS if all partitions were created successfully.

CY_U3P_ERROR_NOT_STARTED if the storage driver has not been started.

CY_U3P_ERROR_INVALID_DEV if the storage device specified does not exist.

CY_U3P_ERROR_ALREADY_PARTITIONED if the storage device has already been partitioned.

CY_U3P_ERROR_INVALID_ADDR if the partition sizes specified cannot be setup.

CY_U3P_ERROR_BAD_ARGUMENT if the partition count or types specified are invalid.

CY_U3P_ERROR_DEVICE_BUSY if the device is busy processing another request.

Parameters

portId	Storage port to do the partitioning on.
partCount	Total number of partitions required. Should be less than or equal to 4.
partSizes_p	Sizes for each of the first (N - 1) partitions. The remaining storage on the device will be assigned to the last partition.
partType_p	Type for each partition (boot or data partition).

5.26.4.24 CyU3PReturnStatus_t CyU3PSibQueryDevice (uint8_t *portId*, CyU3PSibDevInfo_t * *devInfo_p*)

Query storage device information.

Description

This function returns the storage device information for the specified port. If the operation completes successfully the structure will be populated with the storage device information.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_BAD_ARGUMENT if the portId or devInfo_p arguments are bad.

CY_U3P_ERROR_INVALID_DEV if the device is not valid.

See Also

CyU3PSibDevInfo_t

Parameters

portId	The storage device to be queried.
devInfo_p	Pointer to the storage device info structure.

5.26.4.25 CyU3PReturnStatus_t CyU3PSibQueryUnit (uint8_t *portId*, uint8_t *unitId*, CyU3PSibLunInfo_t * *unitInfo_p*)

Query storage partition information.

Description

This function returns the storage partition (LUN) information for the specified port and unit number. If the operation completes successfully, the structure will be populated with the storage unit information.

Return value

CY_U3P_SUCCESS if the operation completes successfully.

CY_U3P_ERROR_INVALID_DEV if the device is not valid.

CY_U3P_ERROR_INVALID_UNIT if the unit is not valid.

See Also

CyU3PSibLunInfo_t

Parameters

portId	The storage device to be queried.
unitId	The storage unit to be queried.
unitInfo_p	Pointer to the storage unit info structure.

5.26.4.26 CyU3PReturnStatus_t CyU3PSibReadWriteRequest (CyBool_t *isRead*, uint8_t *portId*, uint8_t *unitId*, uint16_t *numBlks*, uint32_t *blkAddr*, uint8_t *socket*)

Initiates a read/write data request.

Description

This function is used to initiate a read/write request to a storage device. The open-ended read/write commands (CMD18 and CMD25) are used in all cases. The CY_U3P_SIB_EVENT_XFER_CPLT event will be sent to the caller through the register storage callback function when the specified amount of data has been completely transferred.

This function internally initiates the DMA channel for transferring the specified amount of data. It is therefore expected that the DMA channel is left in the idle state when calling this API.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_BAD_ARGUMENT if the port or unit number is out of range.

CY_U3P_ERROR_INVALID_DEV if the device specified is not present.

CY_U3P_ERROR_CARD_LOCKED if the storage device being accessed is password locked.

CY_U3P_ERROR_WRITE_PROTECTED if the storage device is write protected.

CY_U3P_ERROR_INVALID_UNIT if the LUN specified is not a valid LUN.

CY_U3P_ERROR_INVALID_ADDR if the address specified is not valid.

CY_U3P_ERROR_DEVICE_BUSY if the storage device is already busy processing another request.

See Also

CyU3PSibCommitReadWrite
CyU3PSibSetWriteCommitSize

Parameters

isRead	Read or a write operation
portId	Storage port number to access.
unitId	Storage partition (unit) number to access.
numBlks	Number of blocks to be read or written.
blkAddr	Start address of the operation
socket	The SIB socket on the which the operation is to happen.

5.26.4.27 CyU3PReturnStatus_t CyU3PSibRegisterCbk (CyU3PSibEvtCbk_t *sibEvtCbk*)

Register a function for notification of storage related events.

Description

This function is used to register a callback function which will be called to provide notification of storage related events.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_BAD_ARGUMENT if the parameter is not valid.

See Also

CyU3PSibEvtCbk_t

Parameters

sibEvtCbk	The event callback function to be called.

5.26.4.28 CyU3PReturnStatus_t CyU3PSibRemovePartitions (uint8_t *portId*)

Remove all partitions and re-unify a storage device.

Description

This function is used to remove all partitions created using the CyU3PSibPartitionStorage API, and re-unify the complete storage available on a device as a single data partition (volume).

Return value

CY_U3P_SUCCESS if the partitions were successfully removed.

CY_U3P_ERROR_NOT_STARTED if the storage driver has not been started.

CY_U3P_ERROR_INVALID_DEV if the device specified does not exist.

CY_U3P_ERROR_NOT_PARTITIONED if the device has not been partitioned.

CY_U3P_ERROR_DEVICE_BUSY if the device is busy processing another request.

See Also

CyU3PSibPartitionStorage

Parameters

portId	Storage port to remove partitions on.

5.26.4.29 CyU3PReturnStatus_t CyU3PSibRemovePasswd (uint8_t *portId,* uint8_t * *passwd,* uint8_t *passwdLen*)

Clear the password on an SD Card.

Description

This function is used to clear the password on an SD Card on the specified port.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_CARD_LOCK_UNLOCK if the operation failed while removing password.

Parameters

portId	Port on which the clear password operation is to be done.
passwd	The current password set in the card.
passwdLen	The current password length in bytes.

5.26.4.30 CyU3PReturnStatus_t CyU3PSibSendSwitchCommand (uint8_t *portId*, uint32_t *argument*, uint32_t * *resp_p*)

Send a SWITCH command to update the Ext. CSD register on an MMC device.

Description

This API can be used to send a SWITCH command to update a Ext. CSD register byte on an MMC device. While SD cards support the SWITCH command, their usage model is different; and this API does not support them.

Return value

CY_U3P_SUCCESS if the switch command was sent successfully.

CY_U3P_ERROR_BAD_ARGUMENT if an invalid port ID is specified.

CY_U3P_ERROR_INVALID_DEV if the addressed device is not an MMC device.

CY_U3P_ERROR_DEVICE_BUSY if the MMC device is busy performing data transfer.

See Also

CyU3PSibGetMMCExtCsd

Parameters

portId	Port on which to send the Switch command.
argument	Switch command argument. No validation is done on this parameter.
resp_p	Buffer to read the Switch command response into (optional, can be NULL).

5.26.4.31 void CyU3PSibSetBlockLen (uint8_t *portId*, uint16_t *blkLen*)

Define the block length to be used for storage data transfers.

Description

This function is used to configure the block length to be used for storage data transfers through the FX3S device. Note that this API sets the block length only in the FX3S device registers. If a corresponding SD/MMC command is to be sent to the storage device, that needs to be done separately through the CyU3PSibVendorAccess API.

Return value

None

See Also

CyU3PSibVendorAccess

Parameters

portId	Port id on which to change the block length.
blkLen	Desired block length in bytes.

5.26.4.32 CyU3PReturnStatus_t CyU3PSibSetIntfParams (uint8_t *portId*, CyU3PSibIntfParams_t * *intfParams*)

Define the interface parameters for a storage port.

Description

This function is used to define the interface parameters such as low voltage operation, DDR clocking support, card detection and write protection support for one of the FX3S storage ports. This function can be called before the storage driver is started up, so that these parameters are effective when CyU3PSibStart is called.

Return value

CY_U3P_SUCCESS if the parameters were successfully updated.

CY_U3P_ERROR_BAD_ARGUMENT if the parameters passed are incorrect.

See Also

CyU3PSibStart
CyU3PSibIntfParams_t

Parameters

portId	Storage port to be updated.
intfParams	Parameters to be selected for the storage interface.

5.26.4.33 CyU3PReturnStatus_t CyU3PSibSetPasswd (uint8_t *portId*, CyBool_t *lockCard*, uint8_t * *passwd*, uint8_t *passwdLen*, uint8_t * *newPasswd*, uint8_t *newPasswdLen*)

Set/replace password on an SD Card.

Description

This function is used to set or replace the password on an SD Card on the specified port. The card can also be locked while setting the password by setting the lockCard variable.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_CARD_LOCK_UNLOCK if setting the password failed.

Parameters

portId	Port on which the set/replace password operation is to be done.
lockCard	Specifies whether to lock the card while setting the password.
passwd	The password to be set on the card. In case the password is being replaced this indicates the current password in use.
passwdLen	The length of the password in bytes. In case the password is being replaced this indicates the current password length in bytes.
newPasswd	The new password to be set on the card.
newPasswdLen	The new password length. This should be set to zero in case the password is not being replaced.

5.26.4.34 CyU3PReturnStatus_t CyU3PSibSetWriteCommitSize (uint8_t *portId*, uint32_t *numSectors*)

Set the sector granularity at which write operations are committed to the SD/MMC storage.

Description

Committing write data to SD/MMC storage in small chunks can lead to poor write transfer performance. The write performance to these devices can be improved by combining write operations to sequential addresses, and performing a single commit (STOP_TRANSMISSION) operation.

This function is used to set the granularity at which write operations are committed to the storage device. By default, this value is set to 1, meaning that a write of one sector or more will be immediately committed to the storage device.

Return value

CY_U3P_SUCCESS if the write granularity is set properly.

CY_U3P_ERROR_BAD_ARGUMENT if the portId is invalid or the numSectors value is set to 0.

See Also

CyU3PSibReadWriteRequest
CyU3PSibCommitReadWrite

Parameters

portId	Storage port being configured.
numSectors	Number of sectors after which a write operation should be committed to the SD/MMC storage.

5.26.4.35 CyU3PReturnStatus_t CyU3PSibStart (void)

Start the storage driver and try to initialize any storage devices connected.

Description

This function starts the storage driver on the FX3S device, and initializes any storage devices that are connected on the two storage ports.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_NOT_SUPPORTED if the part in use does not support the SD/MMC interface.

CY_U3P_ERROR_NOT_CONFIGURED if the IO Matrix confiuration is not correct for the storage operation.

CY_U3P_ERROR_SIB_INIT if DMA channel creation associated with the storage port fails.

See Also

CyU3PSibStop
CyU3PSibInit

5.26.4.36 void CyU3PSibStop (void)

Stop the storage driver on the FX3S device.

Description

De-initializes any connected storage devices and stops the storage driver.

Return value

None

See Also

CyU3PSibStart
CyU3PSibDeInit

5.26.4.37 CyU3PReturnStatus_t CyU3PSibVendorAccess (uint8_t *portId*, uint8_t *cmd*, uint32_t *cmdArg*, uint8_t *respLen*, uint8_t * *respData*, uint16_t *numBlks*, CyBool_t *isRead*, uint8_t *socket*)

Vendor SD/MMC command access.

Description

This function is used to send general SD/MMC commands to storage device/cards attached to FX3S device and receive the corresponding response. Custom commands can be implemented using this function.

The response data will be placed in the output parameter respData along with the CRC7 and end bits. respLen parameter indicates the expected response length (in bits) not including the start, transmit, CRC7 and end bits.

If the command being executed has an associated data phase; the numBlks, socket and pChHandle parameters can be used to manage the data transfer.

Return value

CY_U3P_SUCCESS if the operation completed successfully.

CY_U3P_ERROR_BAD_ARGUMENT if the parameters passed to the function are not valid.

CY_U3P_ERROR_TIMEOUT timed out while waiting for the response from the card.

CY_U3P_ERROR_DEVICE_BUSY if the device is busy processing another request.

See Also

CyU3PSibSetBlockLen

Parameters

portId	Storage port on which command is to be executed.
cmd	SD/MMC Commande to be sent
cmdArg	Command argument
respLen	Length of response in bits
respData	Pointer to response data
numBlks	Length of data to be transferred in blocks. Set to 0 if there is no data phase.
isRead	Direction of data transfer, in case there is a data phase.
socket	S-Port socket to be used.

5.27 firmware/u3p_firmware/inc/cyu3socket.h File Reference

All block based data transfers IN or OUT of the FX3 device are performed through hardware blocks called sockets. An end-to-end data flow through the FX3 device (DMA channel) is created by tying two or more sockets together using a shared set of data buffers for intermediate storage. This file provides a set of low level APIs that operate on these sockets, and allow custom data pipes to be created.

```
#include "cyu3types.h"
#include "cyu3descriptor.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PDmaSocket_t

  DMA socket register structure.
- struct CyU3PDmaSocketConfig_t

  DMA socket configuration structure.

Macros

- #define **CY_U3P_DMA_LPP_MIN_CONS_SCK** (0) /∗ The min consumer socket number for Serial IOs. ∗/
- #define **CY_U3P_DMA_LPP_MAX_CONS_SCK** (4) /∗ The max consumer socket number for Serial IOs. ∗/
- #define **CY_U3P_DMA_LPP_MIN_PROD_SCK** (5) /∗ The min producer socket number for Serial IOs. ∗/
- #define **CY_U3P_DMA_LPP_MAX_PROD_SCK** (7) /∗ The max producer socket number for Serial IOs. ∗/
- #define **CY_U3P_DMA_LPP_NUM_SCK** (8) /∗ The number of sockets for Serial IOs. ∗/
- #define **CY_U3P_DMA_PIB_MIN_CONS_SCK** (0) /∗ The min consumer socket number for GPIF-II. ∗/
- #define **CY_U3P_DMA_PIB_MAX_CONS_SCK** (15) /∗ The max consumer socket number for GPIF-II. ∗/

- #define **CY_U3P_DMA_PIB_MIN_PROD_SCK** (0) /∗ The min producer socket number for GPIF-II. ∗/
- #define **CY_U3P_DMA_PIB_MAX_PROD_SCK** (31) /∗ The max producer socket number for GPIF-II. ∗/
- #define **CY_U3P_DMA_PIB_NUM_SCK** (32) /∗ The number of sockets for GPIF-II. ∗/
- #define **CY_U3P_DMA_UIB_MIN_CONS_SCK** (0) /∗ The min consumer socket number for USB egress EPs. ∗/
- #define **CY_U3P_DMA_UIB_MAX_CONS_SCK** (15) /∗ The max consumer socket number for USB egress EPs. ∗/
- #define **CY_U3P_DMA_UIB_NUM_SCK** (16) /∗ The number of sockets for USB egress EPs. ∗/
- #define **CY_U3P_DMA_UIBIN_MIN_PROD_SCK** (0) /∗ The min producer socket number for USB ingress EPs. ∗/
- #define **CY_U3P_DMA_UIBIN_MAX_PROD_SCK** (15) /∗ The max producer socket number for USB ingress EPs. ∗/
- #define **CY_U3P_DMA_UIBIN_NUM_SCK** (16) /∗ The number of sockets for USB ingress EPs. ∗/
- #define **CY_U3P_DMA_CPU_NUM_SCK** (2) /∗ The number of sockets for CPU. ∗/
- #define **CY_U3P_IP_BLOCK_POS** (8) /∗ The ip block position. ∗/
- #define **CY_U3P_IP_BLOCK_MASK** (0x3F) /∗ The ip block mask. ∗/
- #define **CY_U3P_DMA_SCK_MASK** (0xFF) /∗ The DMA socket mask. ∗/
- #define **CY_U3P_DMA_SCK_ID_MASK** (0x3FFF) /∗ The DMA socket id mask. ∗/
- #define CyU3PDmaGetSckNum(sckId) ((sckId) & CY_U3P_DMA_SCK_MASK)

  Get the socket number field from the socket ID.
- #define CyU3PDmaGetIpNum(sckId) (((sckId) >> CY_U3P_IP_BLOCK_POS) & CY_U3P_IP_BLOCK_MASK)

  Get the ip number field from the socket ID.
- #define CyU3PDmaGetSckId(ipNum, sckNum)

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

Typedefs

- typedef enum CyU3PBlockId_t CyU3PBlockId_t

  DMA IP block IDs.
- typedef struct CyU3PDmaSocket_t CyU3PDmaSocket_t

  DMA socket register structure.
- typedef struct CyU3PDmaSocketConfig_t CyU3PDmaSocketConfig_t

  DMA socket configuration structure.
- typedef void(∗ CyU3PDmaSocketCallback_t)(uint16_t sckId, uint32_t status)

  DMA socket interrupt handler callback function.

Enumerations

- enum CyU3PBlockId_t { CY_U3P_LPP_IP_BLOCK_ID = 0, CY_U3P_PIB_IP_BLOCK_ID = 1, CY_U3P_SIB_IP_BLOCK_ID = 2, CY_U3P_UIB_IP_BLOCK_ID = 3, CY_U3P_UIBIN_IP_BLOCK_ID = 4, CY_U3P_NUM_IP_BLOCK_ID = 5, CY_U3P_CPU_IP_BLOCK_ID = 0x3F }

  DMA IP block IDs.

Functions

- CyBool_t CyU3PDmaSocketIsValid (uint16_t sckId)

  Validates the socket ID.
- CyBool_t CyU3PDmaSocketIsValidProducer (uint16_t sckId)

  Check whether the specified socket can be used as a producer.
- CyBool_t CyU3PDmaSocketIsValidConsumer (uint16_t sckId)

  Check whether the specified socket can be used as a consumer.
- CyU3PReturnStatus_t CyU3PDmaSocketSetConfig (uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p)

  Sets the socket configuration.
- CyU3PReturnStatus_t CyU3PDmaSocketGetConfig (uint16_t sckId, CyU3PDmaSocketConfig_t *sck_p)

  Fetch the current socket configuration.
- void CyU3PDmaSocketSetWrapUp (uint16_t sckId)

  Request a socket to wrap up.
- void CyU3PDmaSocketDisable (uint16_t sckId)

  Disable the selected socket.
- void CyU3PDmaSocketEnable (uint16_t sckId)

  Enable the selected socket.
- void CyU3PDmaUpdateSocketSuspendOption (uint16_t sckId, uint16_t suspendOption)

  Update the options for the socket suspend.
- void CyU3PDmaUpdateSocketResume (uint16_t sckId, uint16_t suspendOption)

  Resume a socket from the suspended state.
- void CyU3PDmaSocketSendEvent (uint16_t sckId, uint16_t dscrIndex, CyBool_t isOccupied)

  Send an event to the socket.
- void CyU3PDmaSocketRegisterCallback (CyU3PDmaSocketCallback_t cb)

  Register a callback handler for custom socket interrupts.

5.27.1 Detailed Description

All block based data transfers IN or OUT of the FX3 device are performed through hardware blocks called sockets. An end-to-end data flow through the FX3 device (DMA channel) is created by tying two or more sockets together using a shared set of data buffers for intermediate storage. This file provides a set of low level APIs that operate on these sockets, and allow custom data pipes to be created. **Note**

These functions are primarily used by the FX3 library to implement the DMA channel APIs. It is recommended that the channel APIs be used instead of directly calling these functions.

5.27.2 Macro Definition Documentation

5.27.2.1 #define CyU3PDmaGetSckId(*ipNum, sckNum*)

Value:

```
(((sckNum) & CY_U3P_DMA_SCK_MASK) | \
        (((ipNum) & CY_U3P_IP_BLOCK_MASK) << CY_U3P_IP_BLOCK_POS))
```

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

5.27.3 Typedef Documentation

5.27.3.1 typedef enum CyU3PBlockId_t CyU3PBlockId_t

DMA IP block IDs.

Description

The distributed DMA fabric on the FX3 device identifies sockets based on a two part address. The first part identifies the IP block which implements the socket, and the second part identifies the specific socket within the IP block.

This type lists the various IP blocks on the FX3/FX3S devices that provide DMA sockets.

See Also

CyU3PDmaSocketId_t

5.27.3.2 typedef struct CyU3PDmaSocket_t CyU3PDmaSocket_t

DMA socket register structure.

Description

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

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

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

See Also

CyU3PDmaSocketConfig_t

5.27.3.3 typedef void(∗ CyU3PDmaSocketCallback_t)(uint16_t sckId,uint32_t status)

DMA socket interrupt handler callback function.

Description

When the DMA channel APIs are used for data transfer, all socket specific interrupts are handled by the DMA driver and reported to the user through the DMA channel callbacks.

If a particular socket is being controlled directly using the socket APIs in this file, the driver will be unable to determine how the socket interrupts should be handled. These interrupts are reported to the user application for handling through a callback function of this type.

5.27.3.4 typedef struct CyU3PDmaSocketConfig_t CyU3PDmaSocketConfig_t

DMA socket configuration structure.

Description

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

See Also

CyU3PDmaSocket_t
CyU3PDmaSocketSetConfig
CyU3PDmaSocketGetConfig

5.27.4 Enumeration Type Documentation

5.27.4.1 enum CyU3PBlockId_t

DMA IP block IDs.

Description

The distributed DMA fabric on the FX3 device identifies sockets based on a two part address. The first part identifies the IP block which implements the socket, and the second part identifies the specific socket within the IP block.

This type lists the various IP blocks on the FX3/FX3S devices that provide DMA sockets.

See Also

CyU3PDmaSocketId_t

Enumerator

CY_U3P_LPP_IP_BLOCK_ID LPP block contains UART, I2C, I2S and SPI.

CY_U3P_PIB_IP_BLOCK_ID P-port interface (GPIF) block.

CY_U3P_SIB_IP_BLOCK_ID Storage interface block (FX3S only).

CY_U3P_UIB_IP_BLOCK_ID USB interface block for egress sockets.

CY_U3P_UIBIN_IP_BLOCK_ID USB interface block for ingress sockets.

CY_U3P_NUM_IP_BLOCK_ID Sentinel value. Count of valid DMA IPs.

CY_U3P_CPU_IP_BLOCK_ID Special value used to denote CPU as DMA endpoint.

5.27.5 Function Documentation

5.27.5.1 void CyU3PDmaSocketDisable (uint16_t *sckId*)

Disable the selected socket.

Description

This function disables the specified socket, and returns only after it has been disabled. If the socket is forcibly disabled during data transfer, there will be loss of data.

Return value

None

See Also

CyU3PDmaSocketEnable

Parameters

sckId	Id of socket to be disabled.

5.27.5.2 void CyU3PDmaSocketEnable (uint16_t *sckId*)

Enable the selected socket.

Description

This function enables the specified DMA socket. This should be called only after the configuring the socket with valid values using the CyU3PDmaSocketSetConfig API.

Return value

None

See Also

CyU3PDmaSocketDisable
CyU3PDmaSocketSetConfig

Parameters

sckId	Id of socket to be enabled.

5.27.5.3 CyU3PReturnStatus_t CyU3PDmaSocketGetConfig (uint16_t *sckId*, CyU3PDmaSocketConfig_t * *sck_p*)

Fetch the current socket configuration.

Description

Fetch the current configuration and status of a DMA socket.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if the sck_p pointer in NULL.

See Also

CyU3PDmaSocketSetConfig

Parameters

sckId	Socket id whose configuration is to be retrieved.
sck_p	Output parameter to be filled with the socket configuration.

5.27.5.4 CyBool_t CyU3PDmaSocketIsValid (uint16_t *sckId*)

Validates the socket ID.

Description

The function validates if the given socket id exists on the device. CPU sockets are virtual sockets and so the function considers them as invalid. The function considers a socket invalid if the IP block that implements it has not been started.

Return value

CyTrue if the socket is valid

CyFalse if the socket is invalid.

See Also

CyU3PDmaSocketIsValidProducer
CyU3PDmaSocketIsValidConsumer

Parameters

sckId	Socket id to be validated.

5.27.5.5 CyBool_t CyU3PDmaSocketIsValidConsumer (uint16_t *sckId*)

Check whether the specified socket can be used as a consumer.

Description

Some of the DMA sockets on the FX3 device can only be used as producers or ingress sockets. This function checks whether the specified socket is valid and can be used as a consumer.

Return value

CyTrue if the socket is a valid consumer.

CyFalse if the socket is invalid or is ingress-only.

See Also

CyU3PDmaSocketIsValid
CyU3PDmaSocketIsValidProducer

Parameters

sckId	Socket id to be validated.

5.27.5.6 CyBool_t CyU3PDmaSocketIsValidProducer (uint16_t *sckId*)

Check whether the specified socket can be used as a producer.

Description

Some of the DMA sockets on the FX3 device can only be used as consumers or egress sockets. This function checks whether the specified socket is valid and can be used as a producer.

Return value

CyTrue if the socket is a valid producer.

CyFalse if the socket is invalid or if it is egress-only.

See Also

CyU3PDmaSocketIsValid
CyU3PDmaSocketIsValidConsumer

Parameters

sckId	Socket id to be validated.

5.27.5.7 void CyU3PDmaSocketRegisterCallback (CyU3PDmaSocketCallback_t *cb*)

Register a callback handler for custom socket interrupts.

Description

This function registers a callback function that will be called to notify the user about socket interrupts on a socket that is not associated with any DMA channels.

Return value

None

See Also

CyU3PDmaSocketCallback_t

Parameters

cb	Callback function pointer. Can be zero to unregister previous callback.

5.27.5.8 void CyU3PDmaSocketSendEvent (uint16_t *sckId*, uint16_t *dscrIndex*, CyBool_t *isOccupied*)

Send an event to the socket.

Description

Send a produce / consume event to the selected socket. This API is used to synchronize sockets on manual DMA channels.

Return value

None

See Also

CyU3PDmaSocketResume

Parameters

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

5.27.5.9 CyU3PReturnStatus_t CyU3PDmaSocketSetConfig (uint16_t *sckId*, CyU3PDmaSocketConfig_t * *sck_p*)

Sets the socket configuration.

Description

Updates the configuration for a DMA socket based on the values in the structure provided.

Return value

CY_U3P_SUCCESS - if the function call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if the sck_p pointer in NULL.

See Also

CyU3PDmaSocketConfig_t
CyU3PDmaSocketGetConfig

Parameters

sckId	Socket id whose configuration is to be updated.
sck_p	Pointer to structure containing socket configuration.

5.27.5.10 void CyU3PDmaSocketSetWrapUp (uint16_t *sckId*)

Request a socket to wrap up.

Description

This function sets the bit that forces a socket with a partially filled buffer to wrap up the buffer. The function does not wait for the socket to wrap up.

Note

This API should be called after ensuring that the socket in question is not actively receiving data. Otherwise, this can result in data loss.

Return value

None

See Also

CyU3PDmaSocketEnable
CyU3PDmaSocketDisable

Parameters

sckId	Id of socket to be wrapped up.

5.27.5.11 void CyU3PDmaUpdateSocketResume (uint16_t *sckId*, uint16_t *suspendOption*)

Resume a socket from the suspended state.

Description

This function clears all active suspend conditions on the selected socket and resumes its operation.

Return value

None

See Also

CyU3PDmaUpdateSocketSuspendOption

Parameters

sckId	Id of socket to set the option.
suspendOption	Active suspend options. This should match the suspend options as set by the CyU3PDmaUpdateSocketSuspendOption API.

5.27.5.12 void CyU3PDmaUpdateSocketSuspendOption (uint16_t *sckId*, uint16_t *suspendOption*)

Update the options for the socket suspend.

Description

This API updates the suspend options for the selected socket. This does not actually suspend the socket, and the socket suspension will only happen when the pre-conditions for the selected suspend option have been satisfied.

Return value

None

See Also

CyU3PDmaUpdateSocketResume

Parameters

sckId	Id of socket to set the option.
suspendOption	Suspend option.

5.28 firmware/u3p_firmware/inc/cyu3spi.h File Reference

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. The SPI driver module provides functions to configure the SPI interface parameters and to perform data read/writes from/to the slave devices.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PSpiConfig_t

  Structure defining the configuration of SPI interface.

Macros

- #define CY_U3P_SPI_DEFAULT_LOCK_TIMEOUT (CYU3P_WAIT_FOREVER)

  Delay duration to wait to get a lock on the SPI block.

Typedefs

- typedef enum CyU3PSpiEvt_t CyU3PSpiEvt_t

  List of SPI related event types.
- typedef enum CyU3PSpiError_t CyU3PSpiError_t

  List of SPI specific error / status codes.
- typedef enum CyU3PSpiSsnLagLead_t CyU3PSpiSsnLagLead_t

  Enumeration defining the lead and lag times of SSN with respect to SCK.
- typedef enum CyU3PSpiSsnCtrl_t CyU3PSpiSsnCtrl_t

  Enumeration defining the various ways in which the SSN for a SPI device can be controlled.
- typedef struct CyU3PSpiConfig_t CyU3PSpiConfig_t

  Structure defining the configuration of SPI interface.
- typedef void(∗ CyU3PSpiIntrCb_t)(CyU3PSpiEvt_t evt, CyU3PSpiError_t error)

  Prototype of SPI event callback function.

Enumerations

- enum CyU3PSpiEvt_t { CY_U3P_SPI_EVENT_RX_DONE = 0, CY_U3P_SPI_EVENT_TX_DONE, CY_U3P_SPI_EVENT_ERROR }

  List of SPI related event types.
- enum CyU3PSpiError_t { CY_U3P_SPI_ERROR_TX_OVERFLOW = 12, CY_U3P_SPI_ERROR_RX_UNDERFLOW = 13, CY_U3P_SPI_ERROR_NONE = 15 }

  List of SPI specific error / status codes.

- enum CyU3PSpiSsnLagLead_t {
  CY_U3P_SPI_SSN_LAG_LEAD_ZERO_CLK = 0, CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK, CY_U3P_-
  SPI_SSN_LAG_LEAD_ONE_CLK, CY_U3P_SPI_SSN_LAG_LEAD_ONE_HALF_CLK,
  CY_U3P_SPI_NUM_SSN_LAG_LEAD }

  Enumeration defining the lead and lag times of SSN with respect to SCK.
- enum CyU3PSpiSsnCtrl_t {
  CY_U3P_SPI_SSN_CTRL_FW = 0, CY_U3P_SPI_SSN_CTRL_HW_END_OF_XFER, CY_U3P_SPI_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 }

  Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

Functions

- CyU3PReturnStatus_t CyU3PSpiInit (void)

  Starts the SPI interface block on the device.
- CyU3PReturnStatus_t CyU3PSpiDeInit (void)

  Stops the SPI module.
- CyU3PReturnStatus_t CyU3PSpiSetConfig (CyU3PSpiConfig_t ∗config, CyU3PSpiIntrCb_t cb)

  Configures the SPI interface on the FX3 device.
- CyU3PReturnStatus_t CyU3PSpiSetSsnLine (CyBool_t isHigh)

  Assert or de-assert the SSN Line.
- CyU3PReturnStatus_t CyU3PSpiTransmitWords (uint8_t ∗data, uint32_t byteCount)

  Transmits data word by word over the SPI interface.
- CyU3PReturnStatus_t CyU3PSpiReceiveWords (uint8_t ∗data, uint32_t byteCount)

  Receives data word by word over the SPI interface.
- CyU3PReturnStatus_t CyU3PSpiSetBlockXfer (uint32_t txSize, uint32_t rxSize)

  Initiate SPI data transfers in DMA mode.
- CyU3PReturnStatus_t CyU3PSpiDisableBlockXfer (CyBool_t rxDisable, CyBool_t txDisable)

  This function disables the SPI DMA TX/RX functionality.
- CyU3PReturnStatus_t CyU3PSpiWaitForBlockXfer (CyBool_t isRead)

  Wait until the ongoing SPI DMA transfer is finished.
- void CyU3PRegisterSpiCallBack (CyU3PSpiIntrCb_t spiIntrCb)

  This function registers the callback function for notification of SPI events..

5.28.1 Detailed Description

SPI (Serial Peripheral Interface) is a serial interface defined for inter-device communication. The FX3 device includes a SPI master that can connect to a variety of SPI slave devices and function at various speeds and modes. The SPI driver module provides functions to configure the SPI interface parameters and to perform data read/writes from/to the slave devices.

5.28.2 Typedef Documentation

5.28.2.1 typedef struct CyU3PSpiConfig_t CyU3PSpiConfig_t

Structure defining the configuration of SPI interface.

Description

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

See Also

CyU3PSpiSsnCtrl_t
CyU3PSpiSclkParam_t
CyU3PSpiSetConfig

5.28.2.2 typedef enum CyU3PSpiError_t CyU3PSpiError_t

List of SPI specific error / status codes.

Description

This type lists the various SPI specific error / status codes that are sent to the event callback as event data, when the event type is CY_U3P_SPI_EVENT_ERROR.

See Also

CyU3PSpiEvt_t

5.28.2.3 typedef enum CyU3PSpiEvt_t CyU3PSpiEvt_t

List of SPI related event types.

Description

This enumeration lists the various SPI related event codes that are notified to the user application through an event callback.

See Also

CyU3PSpiError_t
CyU3PSpiIntrCb_t

5.28.2.4 typedef void(∗ CyU3PSpiIntrCb_t)(CyU3PSpiEvt_t evt,CyU3PSpiError_t error)

Prototype of SPI event callback function.

Description

This function type defines a callback to be called when an SPI interrupt has been detected. A function of this type can be registered with the SPI driver as a callback function and will be called whenever an event of interest occurs.

See Also

CyU3PSpiEvt_t
CyU3PSpiError_t
CyU3PRegisterSpiCallBack

5.28.2.5 typedef enum CyU3PSpiSsnCtrl_t CyU3PSpiSsnCtrl_t

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

Description

The SSN line can be controlled by the firmware or the FX3 hardware.

If the SSN is controlled by the firmware API, it will be asserted at the beginning of a transfer and is de-asserted at the end of the transfer. If it is controlled by hardware, the SSN assertion and de-assertion is done in sync with the SPI clock.

In addition to these modes, the SSN can be controlled by the firmware application through a GPIO. In this case, it is the responsibility of the application to drive the line appropriately.

The APIs allow only control of one SSN line. The SSN for any additional slaves must be controlled through GPIOs by the application.

See Also

CyU3PSpiConfig_t
CyU3PSpiSetConfig

5.28.2.6 typedef enum CyU3PSpiSsnLagLead_t CyU3PSpiSsnLagLead_t

Enumeration defining the lead and lag times of SSN with respect to SCK.

Description

The SSN needs to lead the SCK at the beginning of a transaction and SSN needs to lag the SCK at the end of a transfer. This enumeration gives customization allowed for this. This enumeration is required only for hardware controlled SSN.

See Also

CyU3PSpiConfig_t
CyU3PSpiSetConfig

5.28.3 Enumeration Type Documentation

5.28.3.1 enum CyU3PSpiError_t

List of SPI specific error / status codes.

Description

This type lists the various SPI specific error / status codes that are sent to the event callback as event data, when the event type is CY_U3P_SPI_EVENT_ERROR.

See Also

CyU3PSpiEvt_t

Enumerator

CY_U3P_SPI_ERROR_TX_OVERFLOW Write to TX_FIFO when FIFO is full.

CY_U3P_SPI_ERROR_RX_UNDERFLOW Read from RX_FIFO when FIFO is empty.

CY_U3P_SPI_ERROR_NONE No error has happened.

5.28.3.2 enum CyU3PSpiEvt_t

List of SPI related event types.

Description

This enumeration lists the various SPI related event codes that are notified to the user application through an event callback.

See Also

CyU3PSpiError_t
CyU3PSpiIntrCb_t

Enumerator

CY_U3P_SPI_EVENT_RX_DONE Reception is completed

CY_U3P_SPI_EVENT_TX_DONE Transmission is done

CY_U3P_SPI_EVENT_ERROR Error has occurred

5.28.3.3 enum CyU3PSpiSsnCtrl_t

Enumeration defining the various ways in which the SSN for a SPI device can be controlled.

Description

The SSN line can be controlled by the firmware or the FX3 hardware.

If the SSN is controlled by the firmware API, it will be asserted at the beginning of a transfer and is de-asserted at the end of the transfer. If it is controlled by hardware, the SSN assertion and de-assertion is done in sync with the SPI clock.

In addition to these modes, the SSN can be controlled by the firmware application through a GPIO. In this case, it is the responsibility of the application to drive the line appropriately.

The APIs allow only control of one SSN line. The SSN for any additional slaves must be controlled through GPIOs by the application.

See Also

CyU3PSpiConfig_t
CyU3PSpiSetConfig

Enumerator

CY_U3P_SPI_SSN_CTRL_FW SSN is controlled by API and is not at clock boundaries.

CY_U3P_SPI_SSN_CTRL_HW_END_OF_XFER SSN is controlled by hardware and is done in sync with clock. The SSN is asserted at the beginning of a transfer, and de-asserted at the end of a transfer or when no data is available to transmit.

CY_U3P_SPI_SSN_CTRL_HW_EACH_WORD SSN is controlled by the hardware and is done in sync with clock. The SSN is asserted at the beginning of transfer of every word, and de-asserted at the end of the transfer of that word.

CY_U3P_SPI_SSN_CTRL_HW_CPHA_BASED If CPHA is 0, the SSN control is per word; and if CPHA is 1, then the SSN control is per transfer.

CY_U3P_SPI_SSN_CTRL_NONE SSN control is done externally by the firmware application.

CY_U3P_SPI_NUM_SSN_CTRL Number of enumerations.

5.28.3.4 enum CyU3PSpiSsnLagLead_t

Enumeration defining the lead and lag times of SSN with respect to SCK.

Description

The SSN needs to lead the SCK at the beginning of a transaction and SSN needs to lag the SCK at the end of a transfer. This enumeration gives customization allowed for this. This enumeration is required only for hardware controlled SSN.

See Also

CyU3PSpiConfig_t
CyU3PSpiSetConfig

Enumerator

CY_U3P_SPI_SSN_LAG_LEAD_ZERO_CLK SSN is in sync with SCK.

CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK SSN leads / lags SCK by a half clock cycle.

CY_U3P_SPI_SSN_LAG_LEAD_ONE_CLK SSN leads / lags SCK by one clock cycle.

CY_U3P_SPI_SSN_LAG_LEAD_ONE_HALF_CLK SSN leads / lags SCK by one and half clock cycles.

CY_U3P_SPI_NUM_SSN_LAG_LEAD Number of possible values.

5.28.4 Function Documentation

5.28.4.1 void CyU3PRegisterSpiCallBack (CyU3PSpiIntrCb_t *spiIntrCb*)

This function registers the callback function for notification of SPI events..

Description

This function registers a callback function that will be called for notification of SPI interrupts. This can also be done through the CyU3PSpiInit API.

Return value

None

See Also

CyU3PSpiIntrCb_t
CyU3PSpiInit

Parameters

spiIntrCb	Callback function pointer.

5.28.4.2 CyU3PReturnStatus_t CyU3PSpiDeInit (void)

Stops the SPI module.

Description

This function disables and powers off the SPI interface. This function can be used to shut off the interface to save power when it is not in use.

Return value

CY_U3P_SUCCESS - When the de-init is successful.

CY_U3P_ERROR_NOT_STARTED - When the SPI block has not been initialized.

See Also

CyU3PSpiInit

5.28.4.3 CyU3PReturnStatus_t CyU3PSpiDisableBlockXfer (CyBool_t *rxDisable*, CyBool_t *txDisable*)

This function disables the SPI DMA TX/RX functionality.

Description

This function disables DMA transfers through the SPI block in the TX and/or RX directions. It is possible to disable only the RX or the TX operation, and leave the transfer in the other direcion running.

Both TX and RX transfers need to be disabled through this API before register mode transfers can be used.

Return value

CY_U3P_SUCCESS - When the DisableBlockXfer is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or if no DMA transfer has been configured.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiSetConfig
CyU3PSpiSetBlockXfer
CyU3PSpiWaitForBlockXfer

Parameters

rxDisable	CyTrue: Disable TX block if active; CyFalse: Ignore TX block.
txDisable	CyTrue: Disable RX block if active; CyFalse: Ignore RX block.

5.28.4.4 CyU3PReturnStatus_t CyU3PSpiInit (void)

Starts the SPI interface block on the device.

Description

This function powers up the SPI interface block on the device and is expected to be the first SPI API function that is called by the application.

Return value

CY_U3P_SUCCESS - When the init is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been enabled in the IO configuration.

CY_U3P_ERROR_ALREADY_STARTED - When the SPI block has already been initialized.

See Also

CyU3PSpiDeInit
CyU3PSpiSetConfig
CyU3PSpiSetSsnLine
CyU3PSpiTransmitWords
CyU3PSpiReceiveWords
CyU3PSpiSetBlockXfer
CyU3PSpiWaitForBlockXfer
CyU3PSpiDisableBlockXfer

5.28.4.5 CyU3PReturnStatus_t CyU3PSpiReceiveWords (uint8_t * *data*, uint32_t *byteCount*)

Receives data word by word over the SPI interface.

Description

Reads data from the SPI slave in register mode. This API should only be called when there is no active DMA transfer.

Return value

CY_U3P_SUCCESS - When the data transfer is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI block has not been configured or initialized.

CY_U3P_ERROR_NULL_POINTER - When the data pointer is NULL.

CY_U3P_ERROR_ALREADY_STARTED - When a DMA transfer is active.

CY_U3P_ERROR_BAD_ARGUMENT - When a configured word length is invalid.

CY_U3P_ERROR_TIMEOUT - If the data transfer times out.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiSetConfig
CyU3PSpiSetSsnLine
CyU3PSpiTransmitWords
CyU3PSpiSetBlockXfer

Parameters

data	Buffer to read the data into.
byteCount	Amount of data to be read. This should be an integral multiple of the SPI word length aligned to the next byte.

5.28.4.6 CyU3PReturnStatus_t CyU3PSpiSetBlockXfer (uint32_t *txSize*, uint32_t *rxSize*)

Initiate SPI data transfers in DMA mode.

Description

This function switches the SPI block to DMA mode, and initiates the desired read/write transfers.

If the txSize parameter is non-zero, then TX is enabled; and if rxSize parameter is non-zero, then RX is enabled. If both TX and RX are enabled, transfers can only happen while both the SPI producer and SPI consumer sockets are ready for data transfer.

If the receive count is less than the transmit count, the data transmit continues after the scheduled reception is complete. However, if the transmit count is lesser, data reception is stalled at the end of the transmit operation. The SPI transmit transfer needs to disabled before the receive can proceed.

A call to SetBlockXfer has to be followed by a call to DisableBlockXfer, before the SPI block can be used in Register mode.

Return value

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 - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiSetConfig
CyU3PSpiWaitForBlockXfer
CyU3PSpiDisableBlockXfer

Parameters

txSize	Number of words to be transmitted (not bytes)
rxSize	Number of words to be received (not bytes)

5.28.4.7 CyU3PReturnStatus_t CyU3PSpiSetConfig (CyU3PSpiConfig_t * *config*, CyU3PSpiIntrCb_t *cb*)

Configures the SPI interface on the FX3 device.

Description

This function is used to configure the SPI master interface on FX3 based on the desired settings to talk to the slave. This function can be called repeatedly to change the settings, if different settings are to be used to communicate with different slave devices.

If the DMA mode is transfer is used, the DMA channel(s) need to be reset after each fresh CyU3PSpiSetConfig API call.

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 needs to be clocked at 2X the interface bit rate. The clock for the SPI block is derived from the system clock through a frequency divider. As the hardware only supports integer and half dividers for the frequency, the firmware uses the closest approximation to the desired bit rate.

Return value

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.

CY_U3P_ERROR_TIMEOUT - Attempt to clear the SPI block FIFOs timed out.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiConfig_t
CyU3PSpiIntrCb_t
CyU3PSpiInit

Parameters

config	Pointer to the SPI config structure.
cb	Callback for receiving SPI events.

5.28.4.8 CyU3PReturnStatus_t CyU3PSpiSetSsnLine (CyBool_t *isHigh*)

Assert or de-assert the SSN Line.

Description

Asserts or de-asserts the SSN Line for the default slave device. This is possible only if the SSN line control is configured for FW controlled mode.

Return value

CY_U3P_SUCCESS - When the call is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or if SSN is not firmware controlled.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiInit
CyU3PSpiSetConfig
CyU3PSpiTransmitWords
CyU3PSpiReceiveWords
CyU3PSpiSetBlockXfer

Parameters

isHigh	CyFalse: Pull down the SSN line, CyTrue: Pull up the SSN line.

5.28.4.9 CyU3PReturnStatus_t CyU3PSpiTransmitWords (uint8_t * *data*, uint32_t *byteCount*)

Transmits data word by word over the SPI interface.

Description

This function is used to transmit data in register mode. The function can be called only if there is no active DMA transfer on the bus. If CyU3PSpiSetBlockXfer was called, then CyU3PSpiDisableBlockXfer needs to be called.

Return value

CY_U3P_SUCCESS - When the data transfer is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI block has not been configured or initialized.

CY_U3P_ERROR_NULL_POINTER - When the data pointer is NULL.

CY_U3P_ERROR_ALREADY_STARTED - When a DMA transfer is active.

CY_U3P_ERROR_BAD_ARGUMENT - When a configured word length is invalid.

CY_U3P_ERROR_TIMEOUT - If the data transfer times out.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiSetConfig
CyU3PSpiSetSsnLine
CyU3PSpiReceiveWords
CyU3PSpiSetBlockXfer

Parameters

data	Source data pointer. This needs to be padded to nearest byte if the word length is not byte aligned.
byteCount	This needs to be a multiple of the word length aligned to the next byte.

5.28.4.10 CyU3PReturnStatus_t CyU3PSpiWaitForBlockXfer (CyBool_t *isRead*)

Wait until the ongoing SPI DMA transfer is finished.

Description

This function can be used to ensure that a previous SPI transaction has completed, in the case where the callback is not being used.

Note

This function will keep waiting for a transfer complete event until the pre-defined timeout period, if it is invoked before any SPI transfers are requested.

Return value

CY_U3P_SUCCESS - When the WaitForBlockXfer is successful.

CY_U3P_ERROR_NOT_CONFIGURED - When SPI has not been configured or initialized.

CY_U3P_ERROR_TIMEOUT - If the transfer is not completed within the timeout period.

CY_U3P_ERROR_FAILURE - When the operation encounters an error.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the SPI block.

See Also

CyU3PSpiSetConfig
CyU3PSpiSetBlockXfer
CyU3PSpiDisableBlockXfer

Parameters

isRead	CyFalse: Write operation, CyTrue: Read operation.

5.29 firmware/u3p_firmware/inc/cyu3system.h File Reference

This file defines the device initialization and configuration functions associated with the FX3 SDK.

```
#include <cyu3types.h>
#include <cyfx3_api.h>
#include <cyu3dma.h>
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PSysClockConfig_t

  Clock configuration for FX3 CPU, DMA and register access.
- struct CyU3PDebugLog_t

  FX3 debug logger data type.

Macros

- #define CY_U3P_SYS_PPORT_WAKEUP_SRC (0x01)

  Bit mask for selecting the GPIF (P-port) source to wake FX3 from suspend / standby.
- #define CY_U3P_SYS_UART_WAKEUP_SRC (0x02)

  Bit mask for selecting the UART CTS as a source to wake FX3 from suspend / standby.
- #define CY_U3P_SYS_USB_VBUS_WAKEUP_SRC (0x04)

  Bit mask for selecting the VBus signal as a source to wake FX3 from suspend / standby.
- #define CY_U3P_SYS_USB_BUS_ACTVTY_WAKEUP_SRC (0x08)

  Bit mask for selecting USB bus activity as a source to wake FX3 from suspend.
- #define CY_U3P_SYS_USB_OTGID_WAKEUP_SRC (0x10)

  Bit mask for selecting the USB OTG ID pin as a source to wake FX3 from suspend.

- #define CY_U3P_DEBUG_DMA_BUFFER_SIZE (0x100)

  Buffer size used on the DMA channel used to output debug log messages.
- #define CY_U3P_DEBUG_DMA_BUFFER_COUNT (8)

  Buffer count used on the DMA channel used to output debug log messages.

Typedefs

- typedef enum CyU3PLppModule_t CyU3PLppModule_t

  List of Low Performance (Serial) Peripherals supported by the FX3 device.
- typedef enum CyU3PDriveStrengthState_t CyU3PDriveStrengthState_t

  Drive Strength configuration for various FX3 IOs.
- typedef enum CyU3PSysClockSrc_t CyU3PSysClockSrc_t

  Clock source for a peripheral block.
- typedef struct CyU3PSysClockConfig_t CyU3PSysClockConfig_t

  Clock configuration for FX3 CPU, DMA and register access.
- typedef enum CyU3PSysThreadId_t CyU3PSysThreadId_t

  FX3 API library thread information.
- typedef struct CyU3PDebugLog_t CyU3PDebugLog_t

  FX3 debug logger data type.

Enumerations

- enum CyU3PLppModule_t { CY_U3P_LPP_I2C = 0, CY_U3P_LPP_I2S, CY_U3P_LPP_SPI, CY_U3P_LPP_UART, CY_U3P_LPP_GPIO }

  List of Low Performance (Serial) Peripherals supported by the FX3 device.
- enum CyU3PDriveStrengthState_t { CY_U3P_DS_QUARTER_STRENGTH = 0, CY_U3P_DS_HALF_STRENGTH, CY_U3P_DS_THREE_QUARTER_STRENGTH, CY_U3P_DS_FULL_STRENGTH }

  Drive Strength configuration for various FX3 IOs.
- enum CyU3PSysClockSrc_t { CY_U3P_SYS_CLK_BY_16 = 0, CY_U3P_SYS_CLK_BY_4, CY_U3P_SYS_CLK_BY_2, CY_U3P_SYS_CLK, CY_U3P_NUM_CLK_SRC }

  Clock source for a peripheral block.
- enum CyU3PSysThreadId_t { CY_U3P_THREAD_ID_INT = 0, CY_U3P_THREAD_ID_DMA, CY_U3P_THREAD_ID_SYSTEM, CY_U3P_THREAD_ID_PIB, CY_U3P_THREAD_ID_UIB, CY_U3P_THREAD_ID_LPP, CY_U3P_THREAD_ID_DEBUG = 8, CY_U3P_THREAD_ID_LIB_MAX = 15 }

  FX3 API library thread information.

Functions

- void CyU3PFirmwareEntry (void)

  This is the entry routine for the firmware.
- void CyU3PToolChainInit (void)

  This is the normal entry routine provided by the tool-chain.
- CyU3PReturnStatus_t CyU3PDeviceInit (CyU3PSysClockConfig_t *clkCfg)

  This function initializes the device.

- CyU3PReturnStatus_t CyU3PDeviceCacheControl (CyBool_t isICacheEnable, CyBool_t isDCacheEnable, CyBool_t isDmaHandleDCache)

  This function initializes the caches on the ARM926 core.
- void CyFxApplicationDefine (void)

  This is the FX3 application definition function.
- CyU3PReturnStatus_t CyU3PSysGetApiVersion (uint16_t *majorVersion, uint16_t *minorVersion, uint16_t *patchNumer, uint16_t *buildNumer)

  This function returns the API library version.
- CyU3PReturnStatus_t CyU3PDeviceGetSysClkFreq (uint32_t *freq)

  This function returns the current SYS_CLK frequency.
- void CyU3PDeviceReset (CyBool_t isWarmReset)

  This function resets the FX3 device.
- CyU3PReturnStatus_t CyU3PSysEnterSuspendMode (uint16_t wakeupFlags, uint16_t polarity, uint16_t *wakeupSource)

  Places the FX3 device in low power suspend mode.
- CyU3PReturnStatus_t CyU3PSysEnterStandbyMode (uint16_t wakeupFlags, uint16_t polarity, uint8_t *bkp_buff_p)

  Places the FX3 device in low power standby mode.
- void CyU3PSysWatchDogConfigure (CyBool_t enable, uint32_t period)

  Configure the watchdog reset control.
- void CyU3PSysWatchDogClear (void)

  Clear the watchdog timer to prevent a device reset.
- CyU3PReturnStatus_t CyU3PDeviceConfigureIOMatrix (CyU3PIoMatrixConfig_t *cfg_p)

  Configures the IO matrix for the device.
- CyU3PReturnStatus_t CyU3PDeviceGpioOverride (uint8_t gpioId, CyBool_t isSimple)

  This function is used to override an IO as a GPIO.
- CyU3PReturnStatus_t CyU3PDeviceGpioRestore (uint8_t gpioId)

  This function is used to restore IO to normal mode.
- CyU3PReturnStatus_t CyU3PSetSerialIoDriveStrength (CyU3PDriveStrengthState_t driveStrength)

  Set the IO drive strength for UART, SPI and I2S interfaces.
- CyBool_t CyU3PIsGpioSimpleIOConfigured (uint32_t gpioId)

  Check whether a specific pin has been selected as a simple GPIO.
- CyBool_t CyU3PIsGpioComplexIOConfigured (uint32_t gpioId)

  Check whether a specific pin has been selected as a complex GPIO.
- CyBool_t CyU3PIsGpioValid (uint8_t gpioId)

  Check whether a specified GPIO ID is valid on the current FX3 part.
- CyBool_t CyU3PIsLppIOConfigured (CyU3PLppModule_t lppModule)

  Check whether a specific serial peripheral has been enabled in the IO Matrix.
- CyU3PPartNumber_t CyU3PDeviceGetPartNumber (void)

  This function is used to get the part number of the FX3 device in use.
- CyU3PReturnStatus_t CyU3PDebugInit (CyU3PDmaSocketId_t destSckId, uint8_t traceLevel)

  Initialize the firmware logging functionality.
- CyU3PReturnStatus_t CyU3PDebugDeInit (void)

  De-initialize the logging function.
- CyU3PReturnStatus_t CyU3PDebugSysMemInit (uint8_t *buffer, uint16_t size, CyBool_t isWrapAround, uint8_t traceLevel)

  Initializes the SYS_MEM based logger facility.
- CyU3PReturnStatus_t CyU3PDebugSysMemDeInit (void)

  Disables the SYS_MEM logger.
- void CyU3PDebugPreamble (CyBool_t sendPreamble)

  Inhibits the preamble bytes preceding a verbose log message.

- CyU3PReturnStatus_t CyU3PDebugPrint (uint8_t priority, char ∗message,...)

  Free form message logging function.
- CyU3PReturnStatus_t CyU3PDebugStringPrint (uint8_t ∗buffer, uint16_t maxLength, char ∗fmt,...)

  Miniature version of snprintf routine.
- CyU3PReturnStatus_t CyU3PDebugLog (uint8_t priority, uint16_t message, uint32_t parameter)

  Log a codified message.
- CyU3PReturnStatus_t CyU3PDebugLogFlush (void)

  Flush any pending logs out of the internal buffers.
- CyU3PReturnStatus_t CyU3PDebugLogClear (void)

  Drop any pending logs in the internal buffers.
- void CyU3PDebugSetTraceLevel (uint8_t level)

  This function sets the priority threshold above which debug traces will be logged.
- void CyU3PDebugEnable (uint16_t threadMask)

  Enable log messages from specified threads.
- uint16_t CyU3PDebugDisable (void)

  Disable log messages from all library threads.

5.29.1 Detailed Description

This file defines the device initialization and configuration functions associated with the FX3 SDK.

5.29.2 Typedef Documentation

5.29.2.1 typedef struct CyU3PDebugLog_t CyU3PDebugLog_t

FX3 debug logger data type.

Description

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

See Also

CyU3PDebugLog
CyU3PDebugPrint

5.29.2.2 typedef enum CyU3PDriveStrengthState_t CyU3PDriveStrengthState_t

Drive Strength configuration for various FX3 IOs.

Description

IOs on the FX3 device are grouped based on function into multiple interfaces (GPIF, I2C, I2S, SPI, UART). The IO drive strength for each of these groups can be separately selected from this list.

See Also

CyU3PSetPportDriveStrength
CyU3PSetI2cDriveStrength
CyU3PSetGpioDriveStrength
CyU3PSetSerialIoDriveStrength

5.29.2.3 typedef enum CyU3PLppModule_t CyU3PLppModule_t

List of Low Performance (Serial) Peripherals supported by the FX3 device.

Description

This enumeration lists the various serial (low performance) peripherals supported by the FX3 device. This type is used to notify the LPP driver about the initialization of individual peripheral modules.

See Also

CyU3PIsLppIOConfigured

5.29.2.4 typedef struct CyU3PSysClockConfig_t CyU3PSysClockConfig_t

Clock configuration for FX3 CPU, DMA and register access.

Description

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

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

The useStandbyClk parameter specifies whether a 32KHz clock has been supplied on the CLKIN_32 pin of the device. This clock is the standby clock for the device. If this pin is not connected, then the device internally generates a clock from the SYS_CLK.

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

This structure is passed as parameter to CyU3PDeviceInit call.

See Also

CyU3PSysClockSrc_t
CyU3PDeviceInit
CyU3PDeviceGetSysClkFreq

5.29.2.5 typedef enum CyU3PSysClockSrc_t CyU3PSysClockSrc_t

Clock source for a peripheral block.

Description

Each peripheral block on FX3 can take various clock values based on the system clock supplied. All clocks are derived from the SYS_CLK_PLL value which depends on the input clock or crystal connected to the device.

The FX3 device identifies the input clock frequency based on the FSLC pins, and these pins should be configured correctly for proper device operation.

The SYS_CLK_PLL value is 416 MHz if the input clock frequency is 26 MHz or 52 MHz; and it is 384 MHz if the input clock frequency is 19.2 MHz or 38.4 MHz. The SYS_CLK_PLL frequency can be changed to 403.2 MHz from 384 MHz by passing appropriate parameters to the CyU3PDeviceInit function.

See Also

CyU3PSysClockConfig_t
CyU3PDeviceGetSysClkFreq
CyU3PDeviceInit

5.29.2.6 typedef enum CyU3PSysThreadId_t CyU3PSysThreadId_t

FX3 API library thread information.

Description

This enumeration holds the thread IDs used by various FX3 API library threads. The thread ID is defined by the first two characters of the thread name provided during thread creation. For example, the DMA thread name is "01_DMA_THREAD".

Thread IDs 0-15 are reserved by the library. Thread IDs from 16 onwards can be used by FX3 application. All threads created in FX3 application are expected to have the first two characters as the thread ID. Please note that this is a convention used for readability of debug logs output through the CyU3PDebugLog function, and is not enforced otherwise.

See Also

CyU3PDebugEnable
CyU3PDebugDisable
CyU3PDebugLog

5.29.3 Enumeration Type Documentation

5.29.3.1 enum CyU3PDriveStrengthState_t

Drive Strength configuration for various FX3 IOs.

Description

IOs on the FX3 device are grouped based on function into multiple interfaces (GPIF, I2C, I2S, SPI, UART). The IO drive strength for each of these groups can be separately selected from this list.

See Also

CyU3PSetPportDriveStrength
CyU3PSetI2cDriveStrength
CyU3PSetGpioDriveStrength
CyU3PSetSerialIoDriveStrength

Enumerator

CY_U3P_DS_QUARTER_STRENGTH The drive strength is one-fourth the maximum

CY_U3P_DS_HALF_STRENGTH The drive strength is half the maximum

CY_U3P_DS_THREE_QUARTER_STRENGTH The drive strength is three-fourth the maximum

CY_U3P_DS_FULL_STRENGTH The drive strength is the maximum

5.29.3.2 enum CyU3PLppModule_t

List of Low Performance (Serial) Peripherals supported by the FX3 device.

Description

This enumeration lists the various serial (low performance) peripherals supported by the FX3 device. This type is used to notify the LPP driver about the initialization of individual peripheral modules.

See Also

CyU3PIsLppIOConfigured

Enumerator

CY_U3P_LPP_I2C I2C Master Module.

CY_U3P_LPP_I2S I2S Master Module.

CY_U3P_LPP_SPI SPI Master Module.

CY_U3P_LPP_UART UART Module.

CY_U3P_LPP_GPIO GPIO Block.

5.29.3.3 enum CyU3PSysClockSrc_t

Clock source for a peripheral block.

Description

Each peripheral block on FX3 can take various clock values based on the system clock supplied. All clocks are derived from the SYS_CLK_PLL value which depends on the input clock or crystal connected to the device.

The FX3 device identifies the input clock frequency based on the FSLC pins, and these pins should be configured correctly for proper device operation.

The SYS_CLK_PLL value is 416 MHz if the input clock frequency is 26 MHz or 52 MHz; and it is 384 MHz if the input clock frequency is 19.2 MHz or 38.4 MHz. The SYS_CLK_PLL frequency can be changed to 403.2 MHz from 384 MHz by passing appropriate parameters to the CyU3PDeviceInit function.

See Also

CyU3PSysClockConfig_t
CyU3PDeviceGetSysClkFreq
CyU3PDeviceInit

Enumerator

CY_U3P_SYS_CLK_BY_16 SYS_CLK divided by 16.

CY_U3P_SYS_CLK_BY_4 SYS_CLK divided by 4.

CY_U3P_SYS_CLK_BY_2 SYS_CLK divided by 2.

CY_U3P_SYS_CLK SYS_CLK frequency.

CY_U3P_NUM_CLK_SRC Number of clock source enumerations.

5.29.3.4 enum CyU3PSysThreadId_t

FX3 API library thread information.

Description

This enumeration holds the thread IDs used by various FX3 API library threads. The thread ID is defined by the first two characters of the thread name provided during thread creation. For example, the DMA thread name is "01_DMA_THREAD".

Thread IDs 0-15 are reserved by the library. Thread IDs from 16 onwards can be used by FX3 application. All threads created in FX3 application are expected to have the first two characters as the thread ID. Please note that this is a convention used for readability of debug logs output through the CyU3PDebugLog function, and is not enforced otherwise.

See Also

CyU3PDebugEnable
CyU3PDebugDisable
CyU3PDebugLog

Enumerator

CY_U3P_THREAD_ID_INT Interrupt context.

CY_U3P_THREAD_ID_DMA Library DMA module thread.

CY_U3P_THREAD_ID_SYSTEM Library system module thread.

CY_U3P_THREAD_ID_PIB Library p-port module thread.

CY_U3P_THREAD_ID_UIB Library USB module thread.

CY_U3P_THREAD_ID_LPP Library low performance peripheral module thread.

CY_U3P_THREAD_ID_DEBUG Library debug module thread.

CY_U3P_THREAD_ID_LIB_MAX Max library reserved thread ID.

5.29.4 Function Documentation

5.29.4.1 void CyFxApplicationDefine (void)

This is the FX3 application definition function.

Description

This function is called by the FX3 RTOS once all of the OS and driver initialization is completed. The RTOS objects and threads required by the application can be created in this function.

This function needs to be defined by the FX3 application firmware. At-least one thread must be created for meaningful operation. This function should not be explicitly called.

Return value

None

See Also

CyU3PKernelEntry

5.29.4.2 CyU3PReturnStatus_t CyU3PDebugDeInit (void)

De-initialize the logging function.

Description

This function de-initializes the firmware logging function and destroys the DMA channel created to output the logs.

Return value

CY_U3P_SUCCESS - When successful.

CY_U3P_ERROR_NOT_STARTED - When the logger is not started.

See Also

CyU3PDebugInit

5.29.4.3 uint16_t CyU3PDebugDisable (void)

Disable log messages from all library threads.

Description

The API returns the previous threadMask set, so that it can be used for re-enabling the logs.

Return value

The previous threadMask value.

See Also

CyU3PDebugEnable

5.29.4.4 void CyU3PDebugEnable (uint16_t *threadMask*)

Enable log messages from specified threads.

Description

Thread ID of 0 means interrupt context. Only threadIds 1 - 15, and interrupt context can be controlled by this API call. This call is intended for only controlling logs from library threads. By default, logs from all library threads are disabled.

Return value

None

See Also

CyU3PDebugDisable

Parameters

threadMask	ThreadID mask whose logs are to be enabled. 1 means enabled.

5.29.4.5 CyU3PReturnStatus_t CyU3PDebugInit (CyU3PDmaSocketId_t *destSckId*, uint8_t *traceLevel*)

Initialize the firmware logging functionality.

Description

This function initializes the firmware logging functionality and creates a DMA channel to send the log traces out through the selected DMA socket.

Return value

CY_U3P_SUCCESS - when successfully initialized.

CY_U3P_ERROR_ALREADY_STARTED - when the logger is already initialized.

CY_U3P_ERROR_BAD_ARGUMENT - when bad socket ID is passed in as parameter.

Other DMA specific error codes are also returned.

See Also

CyU3PDebugDeInit

Parameters

destSckId	Socket through which the logs are to be output.
traceLevel	Priority level beyond which logs should be output.

5.29.4.6 CyU3PReturnStatus_t CyU3PDebugLog (uint8_t *priority*, uint16_t *message*, uint32_t *parameter*)

Log a codified message.

Description

This function is used to output a codified log message which contains a two byte message ID and a four byte parameter. The message ID is expected to be in the range 0x0000 to 0xFFFE.

The log messages are written to a log buffer. If CyU3PDebugSysMemInit has been used, the logs can be cleared using the CyU3PDebugLogClear function.

If CyU3PDebugInit has been used, the log buffer will be written out to the debug socket when it is full. The CyU3P-

DebugLogFlush can be used to flush the contents of the log buffer to the debug socket. The CyU3PDebugLogClear can be used to drop the current contents of the log buffer.

Return value

CY_U3P_SUCCESS - if the Debug log is successful.

CY_U3P_ERROR_NOT_STARTED - if the debug module has not been initialized.

CY_U3P_ERROR_FAILURE - if the logging fails.

See Also

CyU3PDebugInit
CyU3PDebugSysMemInit
CyU3PDebugLogFlush
CyU3PDebugLogClear

Parameters

priority	Priority level for the message.
message	Message ID for the message.
parameter	Parameter associated with the message.

5.29.4.7 CyU3PReturnStatus_t CyU3PDebugLogClear (void)

Drop any pending logs in the internal buffers.

Description

This function is used to ask the logger to drop any pending logs in its internal buffers.

Return value

CY_U3P_SUCCESS - if the Debug log flush is successful.

CY_U3P_ERROR_NOT_STARTED - if the debug module has not been initialized.

CY_U3P_ERROR_INVALID_CALLER - if called from an interrupt.

See Also

CyU3PDebugLog
CyU3PDebugLogFlush

5.29.4.8 CyU3PReturnStatus_t CyU3PDebugLogFlush (void)

Flush any pending logs out of the internal buffers.

Description

All log messages are collected into internal memory buffers on the FX3 device, and then sent out to the target when the buffer is full or when a verbose message is committed.

This function is used to force the logger to send out any pending log messages to the target and flush its internal buffers.

Return value

CY_U3P_SUCCESS - if the Debug log flush is successful.

CY_U3P_ERROR_NOT_STARTED - if the debug module has not been initialized.

CY_U3P_ERROR_INVALID_CALLER - if called from an interrupt.

See Also

CyU3PDebugLog
CyU3PDebugLogClear

5.29.4.9 void CyU3PDebugPreamble (CyBool_t *sendPreamble*)

Inhibits the preamble bytes preceding a verbose log message.

Description

The CyU3PDebugPrint function internally sends 8 byte preamble data preceding the actual message. This preamble data can be used to identify the source and context information for the message.

This function 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).

Return value

None

See Also

CyU3PDebugPrint

Parameters

sendPreamble	CyFalse: Do not send preamble; CyTrue: Send preamble.

5.29.4.10 CyU3PReturnStatus_t CyU3PDebugPrint (uint8_t *priority*, char * *message*, ...)

Free form message logging function.

Description

This function can be used by the application code to log free form message strings, and causes the output message to be written to the UART immediately.

The function supports a subset of the output conversion specifications supported by the printf function. The 'c', 'd', 'u', 'x' and 's' conversion specifications are supported. There is no support for float or double formats, or for flags, precision or type modifiers.

Note

If the full functionality supported by printf such as all conversion specifications, flags, precision, type modifiers etc. are required; please use the standard C library functions to print the formatted message into a string, and then use this function to send the string out through the UART port.

Return value

CY_U3P_SUCCESS - if the Debug print is successful.

CY_U3P_ERROR_NOT_STARTED - if the debug module has not been initialized.

CY_U3P_ERROR_INVALID_CALLER - if called from an interrupt.

CY_U3P_ERROR_BAD_ARGUMENT - if the total length of the formatted string exceeds the debug buffer size.

See Also

CyU3PDebugInit
CyU3PDebugPreamble

Parameters

priority	Priority level for this message.
message	Format specifier string.

5.29.4.11 void CyU3PDebugSetTraceLevel (uint8_t *level*)

This function sets the priority threshold above which debug traces will be logged.

Description

The logger can suppress log messages that have a priority level lower than a user specified threshold. This function is used to set the priority threshold below which logs will be suppressed.

Return value

None

See Also

CyU3PDebugPrint
CyU3PDebugLog

Parameters

level	Lowest debug trace priority level that is enabled.

5.29.4.12 CyU3PReturnStatus_t CyU3PDebugStringPrint (uint8_t * *buffer*, uint16_t *maxLength*, char * *fmt*, ...)

Miniature version of snprintf routine.

Description

This is a mini version of the snprintf routine, which prints formatted debug information into a user provided buffer. The constraints mentioned for the CyU3PDebugPrint function apply for this function as well.

This function only supports the 'c', 'd', 'u', 'x' and 's' conversion specifications. It also does not support precision or type modifiers.

Return value

CY_U3P_SUCCESS - if the print to buffer is successful. CY_U3P_ERROR_BAD_ARGUMENT - if the total length of the formatted string exceeds maxLength.

See Also

CyU3PDebugPrint

Parameters

buffer	Buffer into which the data is to be printed.
maxLength	Buffer size.
fmt	Format specifier string.

5.29.4.13 CyU3PReturnStatus_t CyU3PDebugSysMemDeInit (void)

Disables the SYS_MEM logger.

Description

The function de-initializes the SYS_MEM logger. The normal logger can now be initialized using CyU3PDebugInit call.

Return value

CY_U3P_SUCCESS - when successfully initialized.

CY_U3P_ERROR_NOT_STARTED - when the logger is not yet started.

See Also

CyU3PDebugSysMemInit

5.29.4.14 CyU3PReturnStatus_t CyU3PDebugSysMemInit (uint8_t * *buffer*, uint16_t *size*, CyBool_t *isWrapAround*, uint8_t *traceLevel*)

Initializes the SYS_MEM based logger facility.

Description

This function cannot be used when the CyU3PDebugInit is invoked. The SYS_MEM logging is faster as the writes are done directly to the system memory. But the limitation is that only CyU3PDebugLog function can be used. The log buffer can be cleared by calling CyU3PDebugLogClear.

Return value

CY_U3P_SUCCESS - when successfully initialized.

CY_U3P_ERROR_ALREADY_STARTED - The logger is already started.

CY_U3P_ERROR_NULL_POINTER - If NULL buffer pointer is passed as argument.

CY_U3P_ERROR_BAD_ARGUMENT - If any of the arguments passed is invalid.

See Also

CyU3PDebugSysMemDeInit
CyU3PDebugLog
CyU3PDebugLogClear

Parameters

buffer	The buffer memory to be used for writing the log.
size	The size of memory available for logging.
isWrapAround	CyTrue - wrap up and start logging from beginning; CyFalse - Stop logging on reaching the last memory address.
traceLevel	Priority level beyond which logs should be output.

5.29.4.15 CyU3PReturnStatus_t CyU3PDeviceCacheControl (CyBool_t *isICacheEnable*, CyBool_t *isDCacheEnable*, CyBool_t *isDmaHandleDCache*)

This function initializes the caches on the ARM926 core.

Description

The function is expected to be called immediately after the CyU3PDeviceInit API. The function controls the cache handling in the system. By default (if this API is not called), both Instruction and Data caches are disabled.

It should be noted that once D-cache is enabled, all buffers used for DMA in the system has to be cache line aligned. This means that all DMA buffers have to be 32 byte aligned and 32 byte multiples. This is ensured by the CyU3P-DmaBufferAlloc function. Any buffer allocated outside of this function has to follow the 32 byte alignment / multiple rule. This rule is also applicable for all DMA buffer pointers passed to library APIs including the USB descriptor

buffers assigned using CyU3PUsbSetDesc, data pointers provided to CyU3PUsbSendEP0Data, CyU3PUsbGetEP0Data, CyU3PUsbHostSendSetupRqt etc.

The isDmaHandleDCache determines whether the DMA APIs perform cache cleans and cache flushes internally. If this is CyFalse, then all cache cleans and flushes for buffers used with DMA APIs have to be done explicitly by the user. If this is CyTrue, then the DMA APIs will clean the cache lines for buffers used to send out data from the device and will flush the cache lines for buffers used to receive data.

It is recommended that the isDmaHandleDCache parameter be set to CyTrue if the Data cache is being enabled.

Return value

CY_U3P_SUCCESS - If the call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if isDmaHandleDCache is set to true without enabling the D-cache.

See Also

CyU3PDeviceInit

Parameters

isICacheEnable	Whether to enable the I-cache.
isDCacheEnable	Whether to enable the D-cache.
isDmaHandleDCache	Whether the DMA APIs should internally take care of cache handling.

5.29.4.16 CyU3PReturnStatus_t CyU3PDeviceConfigureIOMatrix (CyU3PIoMatrixConfig_t * *cfg_p*)

Configures the IO matrix for the device.

Description

The FX3/FX3S devices have upto 61 IO pins that can be configured to function in a variety of modes. The functional mode for each of these pins (or groups of pins) should be set based on the desired system level functionality.

The processor port (P-port) of the device can be configured as a GPIF interface. The mode selection for this interface should be based on the means of connecting the external device / processor. Some pins may not be used depending upon the configuration used. The free pins can be used as GPIOs.

If a 16 bit GPIF interface is used, then the DQ[15:0], CTL[4:0] and PMODE pins are reserved for their default usage. CTL[12:5] are not locked and can be configured as GPIO. If any of the CTL[12:5] lines are used for GPIF interface they should not be configured as GPIOs. Similarly if some lines of CTL[4:0] are not used for GPIF they can be overridden to be GPIOs using the CyU3PDeviceGpioOverride call.

If 32 bit GPIF interface is used, all of DQ[31:0] is reserved for the GPIF interface.

Some devices in the family, such as the CYUSB3011 and CYUSB3013 devices, as well as the FX3S device; support only a 16 bit wide GPIF interface. The isDQ32Bit field should be set to false when using these devices.

The FX3S device can support upto two storage ports that can be connected to SD/MMC/SDIO peripherals. The first storage port (S0) is always available and can be configured in a 4 bit SD mode or in a 8 bit MMC mode. The second storage port (S1) shares pins with the serial peripheral interfaces (UART, SPI and I2S).

The serial peripheral interfaces have various configurations that can be used depending on the type of interfaces used. The default mode of operation has all the low performance (serial) peripherals enabled. I2C lines are not multiplexed and are available in all configurations except when used as GPIOs. If a peripheral is marked as not used (CyFalse), then those IO lines can be configured as GPIO.

IOs that are not configured for peripheral interfaces can be used as GPIOs. This selection has to be explicitly made. Otherwise these lines will be configured as per their default function.

The GPIOs available are further classified as simple (set / get a value or receive a GPIO interrupt); or as complex (PWM, timer, counter etc). The selection of this is made via four 32-bit bitmasks where each IO is represented with (1 << IO number).

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

It is recommended that this function be called immediately after the CyU3PDeviceInit call from the main () function; and that the IO matrix not be dynamically changed. However, this API can be invoked when the peripherals affected are disabled.

Return value

CY_U3P_SUCCESS - when the IO configuration is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if some configuration value is invalid.

CY_U3P_ERROR_NOT_SUPPORTED - if the part in use does not support any of the selected features.

See Also

CyU3PIoMatrixConfig_t
CyU3PDeviceGpioOverride
CyU3PDeviceGpioRestore

Parameters

cfg_p	Pointer to Configuration parameters.

5.29.4.17 CyU3PPartNumber_t CyU3PDeviceGetPartNumber (void)

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

Description

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

Return value

Part number of the FX3 device in use.

See Also

CyU3PPartNumber_t

5.29.4.18 CyU3PReturnStatus_t CyU3PDeviceGetSysClkFreq (uint32_t * *freq*)

This function returns the current SYS_CLK frequency.

Description

This function can be used to identify the current SYS_CLK frequency on the FX3 device. This should only be called after the CyU3PDeviceInit API has been called.

Return value

CY_U3P_SUCCESS - If the call succeeds.

CY_U3P_ERROR_NULL_POINTER - if NULL pointer was passed as parameter.

CY_U3P_ERROR_NOT_CONFIGURED - if the device is not initialized.

See Also

CyU3PSysClockConfig_t

Parameters

freq	Pointer to return the current SYS_CLK frequency.

5.29.4.19 CyU3PReturnStatus_t CyU3PDeviceGpioOverride (uint8_t *gpioId,* CyBool_t *isSimple*)

This function is used to override an IO as a GPIO.

Description

This is an override mechanism and can be used to enable any IO line as simple / complex GPIO. This should be done with caution as a wrong setting can cause damage to hardware.

The API is expected to be invoked before the corresponding peripheral module is enabled. The specific GPIO configuration has to be still done after this call.

The CyU3PDeviceConfigureIOMatrix API checks for validity of the configuration, whereas this API does not. Use this API with great caution.

Return value

CY_U3P_SUCCESS if the operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT if the GPIO specified is invalid.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PDeviceGpioRestore

Parameters

gpioId	The IO to be converted to GPIO.
isSimple	CyTrue: simple GPIO, CyFalse: complex GPIO

5.29.4.20 CyU3PReturnStatus_t CyU3PDeviceGpioRestore (uint8_t *gpioId*)

This function is used to restore IO to normal mode.

Description

IOs that are overridden to be GPIO can be restored to normal mode of operation by invoking this API.

Return value

CY_U3P_SUCCESS if the operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT if the GPIO specified is invalid.

See Also

CyU3PDeviceGpioOverride
CyU3PDeviceConfigureIOMatrix

Parameters

gpioId	The GPIO to be restored.

5.29.4.21 CyU3PReturnStatus_t CyU3PDeviceInit (CyU3PSysClockConfig_t * *clkCfg*)

This function initializes the device.

Description

This function initializes the FX3 device by setting up the clocks for the CPU, DMA and register accesses; and then initializing the Vectored Interrupt Controller (VIC). This function also restores the state of the GPIO block if it is called as part of the resumption from low power standby mode.

This function is expected to be the first one called from the main () function, and should be called only once.

Return value

CY_U3P_SUCCESS - If the call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if the parameters are invalid.

See Also

CyU3PSysClockConfig_t

Parameters

clkCfg	The clock configuration for CPU, DMA and register (MMIO) access. The default configuration (CPU divider = 2, DMA divider = 2, MMIO divider = 2, setSysClk400 = CyFalse, useStandbyClk = CyTrue) is selected if the clkCfg parameter is set to NULL.

5.29.4.22 void CyU3PDeviceReset (CyBool_t *isWarmReset*)

This function resets the FX3 device.

Description

This function is used to reset the FX3 device as a whole. Only a whole system reset is supported, as a CPU only reset will leave hardware blocks within the device in an inconsistent state.

If the warm boot option is selected, the firmware in the FX3 RAM shall be maintained; otherwise it shall be discarded and freshly loaded. If the warm boot option is used, the firmware should be capable of initializing any global variables explicitly.

Note

It is recommended that all of the FX3 blocks are de-initialized before triggering a warm boot operation.

Return value

None as this function does not return.

Parameters

isWarmReset	Whether this should be a warm reset or a cold reset.

5.29.4.23 void CyU3PFirmwareEntry (void)

This is the entry routine for the firmware.

Description

This function should be set as the entry routine for the firmware by choosing the appropriate linker settings. This function is defined by the FX3 API library and sets up the runtime stacks that are required for the proper C/C++ language code to execute.

Once all of the initial CPU/memory initialization is completed, this function will call the user define CyU3PToolChainInit function to return control to the user application.

Return value

None

See Also

CyU3PToolChainInit

5.29.4.24 CyBool_t CyU3PIsGpioComplexIOConfigured (uint32_t *gpioId*)

Check whether a specific pin has been selected as a complex GPIO.

Description

This function checks whether a specific FX3 pin has been selected to function as a complex GPIO.

Return value

CyTrue if the pin is selected as a complex GPIO.

CyFalse is the pin is not selected as a complex GPIO.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PIsGpioSimpleIOConfigured

Parameters

gpioId	Pin number to be queried.

5.29.4.25 CyBool_t CyU3PIsGpioSimpleIOConfigured (uint32_t *gpioId*)

Check whether a specific pin has been selected as a simple GPIO.

Description

This function checks whether a specific FX3 pin has been selected to function as a simple GPIO.

Return value

CyTrue if the pin is selected as a simple GPIO.

CyFalse is the pin is not selected as a simple GPIO.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PIsGpioComplexIOConfigured

Parameters

gpioId	Pin number to be queried.

5.29.4.26 CyBool_t CyU3PIsGpioValid (uint8_t *gpioId*)

Check whether a specified GPIO ID is valid on the current FX3 part.

Description

Different parts in the FX3 family support different numbers of GPIOs. This function is used to check whether a specific GPIO number is valid on the current part.

Return value

CyTrue if the GPIO number is valid and usable.

CyFalse if the GPIO number is invalid.

Parameters

gpioId	GPIO number to be checked for validity.

5.29.4.27 CyBool_t CyU3PIsLppIOConfigured (CyU3PLppModule_t *lppModule*)

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

Description

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

Return value

CyTrue if the specified LPP block is enabled.

CyFalse if the specified LPP block is not enabled.

See Also

CyU3PDeviceConfigureIOMatrix
CyU3PLppModule_t

Parameters

lppModule	Serial interface to be queried.

5.29.4.28 CyU3PReturnStatus_t CyU3PSetSerialIoDriveStrength (CyU3PDriveStrengthState_t *driveStrength*)

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

Description

This function is used to update the drive strength for the UART, SPI and I2S interface signals. This is set to CY_U3P_DS_THREE_QUARTER_STRENGTH by default.

Return value

CY_U3P_SUCCESS - If the operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT - If the Drive strength requested is invalid.

See Also

CyU3PDriveStrengthState_t
CyU3PSetI2cDriveStrength

Parameters

driveStrength	Desired serial IO drive strength.

5.29.4.29 CyU3PReturnStatus_t CyU3PSysEnterStandbyMode (uint16_t *wakeupFlags*, uint16_t *polarity*, uint8_t * *bkp_buff_p*)

Places the FX3 device in low power standby mode.

Description

This function places the FX3 device in low power standby mode where the power consumption on the device is lowest. As the power to all the device blocks is removed when in standby mode; this function can only be called after shutting down (de-initializing) all FX3 blocks such as USB, GPIF, UART, I2C etc. Only the GPIO block can be left on, and its state will be restored when the device wakes up.

On wakeup from standby, the device starts firmware execution from the original entry point. In this sense, the entry into and wakeup from standby mode is similar to a warm reset being applied to the FX3 device.

Only P-Port and VBus based wakeup sources are supported to trigger a wake up of the FX3 device from standby mode. While the FX3 System RAM retains its content, the TCM blocks lose power while the device is in standby. This API backs up the content of the I-TCM region into a user specified buffer location before entering standby mode. The I-TCM contents are then automatically restored during re-initialization of the FX3 device.

Return value

No return if the device is actually placed in standby mode.

CY_U3P_ERROR_BAD_ARGUMENT if the wakeup sources, polarity or buffer pointer provided is invalid.

CY_U3P_ERROR_INVALID_SEQUENCE if this API is called while any of the FX3 blocks is still active.

CY_U3P_ERROR_STANDBY_FAILED if the specified wakeup sources are already active.

See Also

CyU3PSysEnterSuspendMode

Parameters

wakeupFlags	List of selected wakeup sources from standby.
polarity	Polarity of the wakeup source which causes the device to wakeup.
bkp_buff_p	Pointer to buffer where the I-TCM content can be backed up. Should be located in the SYSMEM and be able to hold 18 KB of data.

5.29.4.30 CyU3PReturnStatus_t CyU3PSysEnterSuspendMode (uint16_t *wakeupFlags*, uint16_t *polarity*, uint16_t * *wakeupSource*)

Places the FX3 device in low power suspend mode.

Description

The function can be called only after initializing the device completely. The device will enter into the suspended mode until any of the wakeup sources are triggered. This function does not return until the device has already resumed normal operation.

The CPU stops running and the device enters a low power state. Any combination of CY_U3P_SYS_PPORT_WAKEUP_SRC_EN, CY_U3P_SYS_USB_WAKEUP_SRC_EN and CY_U3P_SYS_UART_WAKEUP_SRC_EN can be used as the wakeup trigger.

This function does not affect the state of the USB PHY on the FX3 device. If the USB PHY is to be powered off while in the low power mode, the caller should explicitly call the CyU3PConnectState API to do this.

Return value

CY_U3P_SUCCESS - if the call was successful.

CY_U3P_ERROR_INVALID_CALLER - if called from interrupt context.

CY_U3P_ERROR_BAD_ARGUMENT - if the wakeup sources specified are invalid.

CY_U3P_ERROR_ABORTED - if one of the wakeup source is already triggering.

See Also

CyU3PSysEnterStandbyMode

Parameters

wakeupFlags	Bit mask representing the wakeup sources that are allowed to bring FX3 out of suspend mode.
polarity	Polarity for each of the Wakeup Sources. This field is valid only for the CY_U3P_SYS_UART-_WAKEUP_SRC and CY_U3P_SYS_USB_VBUS_WAKEUP_SRC wakeup sources. 0 - Wakeup when the corresponding source goes low. 1 - Wakeup when the corresponding source goes high.
wakeupSource	Output parameter indicating the source responsible for waking the FX3 from the Suspend mode.

5.29.4.31 CyU3PReturnStatus_t CyU3PSysGetApiVersion (uint16_t * *majorVersion*, uint16_t * *minorVersion*, uint16_t * *patchNumer*, uint16_t * *buildNumer*)

This function returns the API library version.

Description

This function is used to identify the version information for the FX3 API library. It is expected that the cyfxversion.h file corresponding to the actual library version will be used by the application.

Valid pointers need to be passed for each of the return parameters, only if the corresponding return value is required. Otherwise, NULL can be passed.

Return value

CY_U3P_SUCCESS - If the call is successful.

Parameters

majorVersion	Major version number for the release
minorVersion	Minor version number for the release
patchNumer	Patch version for the release
buildNumer	The build number for the release

5.29.4.32 void CyU3PSysWatchDogClear (void)

Clear the watchdog timer to prevent a device reset.

Description

This function is used to clear the watchdog timer so as to prevent it from resetting the FX3 device. This function should be called more frequently than the specified watchdog timer period to avoid unexpected resets.

Return value

None

See Also

CyU3PSysWatchDogConfigure

5.29.4.33 void CyU3PSysWatchDogConfigure (CyBool_t *enable*, uint32_t *period*)

Configure the watchdog reset control.

Description

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

Return value

None

See Also

CyU3PSysWatchDogClear

Parameters

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

5.29.4.34 void CyU3PToolChainInit (void)

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

Description

This routine maps to the start-up code created by the compiler tool-chain. The FX3 firmware will jump to this function after completing the initial CPU, memory and device initialization.

The FX3 SDK includes a version of this function for the GNU ARM tool-chain. This implementation only clears the BSS segment and then jumps to the main function.

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

Return value

None

See Also

CyU3PFirmwareEntry

5.30 firmware/u3p_firmware/inc/cyu3types.h File Reference

This file contains definitions for the basic data types used by the FX3 firmware library. If a compiler provided version is available, this can be used instead of the custom ones, by disabling the CYU3_DEFINE_BASIC_TYPES definition.

```
#include <stdint.h>
```

Macros

- #define CyTrue (1)
- #define CyFalse (0)

Typedefs

- typedef volatile unsigned char uvint8_t
- typedef volatile unsigned short uvint16_t
- typedef volatile unsigned long uvint32_t
- typedef int CyBool_t
- typedef unsigned int CyU3PReturnStatus_t

  Return status from FX3 APIs.

5.30.1 Detailed Description

This file contains definitions for the basic data types used by the FX3 firmware library. If a compiler provided version is available, this can be used instead of the custom ones, by disabling the CYU3_DEFINE_BASIC_TYPES definition.

5.30.2 Macro Definition Documentation

5.30.2.1 #define CyFalse (0)

False value.

5.30.2.2 #define CyTrue (1)

Truth value.

5.30.3 Typedef Documentation

5.30.3.1 typedef int CyBool_t

Boolean data type.

5.30.3.2 typedef unsigned int CyU3PReturnStatus_t

Return status from FX3 APIs.

Description

This is a custom data type used for return status from all FX3 APIs. The specific error codes that can be returned by these APIs are listed in cyu3error.h.

See Also

CyU3PErrorCode_t

5.30.3.3 typedef volatile unsigned short uvint16_t

Volatile 16-bit unsigned value. This is not used in the library.

5.30.3.4 typedef volatile unsigned long uvint32_t

```
Volatile 32-bit unsigned value. Used to represent FX3 device
```

registers.

5.30.3.5 typedef volatile unsigned char uvint8_t

Volatile 8-bit unsigned value. This is not used in the library.

5.31 firmware/u3p_firmware/inc/cyu3uart.h File Reference

The UART driver in FX3 firmware is responsible for handling data transfers through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

```
#include "cyu3types.h"
#include "cyu3lpp.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PUartConfig_t

  Configuration parameters for the UART interface.

Macros

- #define CY_U3P_UART_DEFAULT_LOCK_TIMEOUT (CYU3P_WAIT_FOREVER)

  Delay duration to wait to get a lock on the UART block.

Typedefs

- typedef enum CyU3PUartEvt_t CyU3PUartEvt_t

  List of UART related event types.
- typedef enum CyU3PUartError_t CyU3PUartError_t

  List of UART specific error/status codes.
- typedef enum CyU3PUartBaudrate_t CyU3PUartBaudrate_t

  List of baud rates supported by the UART.
- typedef enum CyU3PUartStopBit_t CyU3PUartStopBit_t

  List of number of stop bits to be used in UART communication.
- typedef enum CyU3PUartParity_t CyU3PUartParity_t

  List of parity settings supported by the UART interface.
- typedef struct CyU3PUartConfig_t CyU3PUartConfig_t

  Configuration parameters for the UART interface.
- typedef void(∗ CyU3PUartIntrCb_t)(CyU3PUartEvt_t evt, CyU3PUartError_t error)

  Prototype of UART event callback function.

Enumerations

- enum CyU3PUartEvt_t { CY_U3P_UART_EVENT_RX_DONE = 0, CY_U3P_UART_EVENT_TX_DONE, CY_U3P_UART_EVENT_ERROR }

  List of UART related event types.
- enum CyU3PUartError_t { CY_U3P_UART_ERROR_NAK_BYTE_0 = 0, CY_U3P_UART_ERROR_RX_PARITY_ERROR = 1, CY_U3P_UART_ERROR_TX_OVERFLOW = 12, CY_U3P_UART_ERROR_RX_UNDERFLOW = 13, CY_U3P_UART_ERROR_RX_OVERFLOW = 14 }

  List of UART specific error/status codes.

- enum CyU3PUartBaudrate_t {
  CY_U3P_UART_BAUDRATE_100 = 100, CY_U3P_UART_BAUDRATE_300 = 300, CY_U3P_UART_BAUDRATE_600 = 600, CY_U3P_UART_BAUDRATE_1200 = 1200,
  CY_U3P_UART_BAUDRATE_2400 = 2400, CY_U3P_UART_BAUDRATE_4800 = 4800, CY_U3P_UART_BAUDRATE_9600 = 9600, CY_U3P_UART_BAUDRATE_10000 = 10000,
  CY_U3P_UART_BAUDRATE_14400 = 14400, CY_U3P_UART_BAUDRATE_19200 = 19200, CY_U3P_UART_BAUDRATE_38400 = 38400, CY_U3P_UART_BAUDRATE_50000 = 50000,
  CY_U3P_UART_BAUDRATE_57600 = 57600, CY_U3P_UART_BAUDRATE_75000 = 75000, CY_U3P_UART_BAUDRATE_100000 = 100000, CY_U3P_UART_BAUDRATE_115200 = 115200,
  CY_U3P_UART_BAUDRATE_153600 = 153600, CY_U3P_UART_BAUDRATE_200000 = 200000, CY_U3P_UART_BAUDRATE_225000 = 225000, CY_U3P_UART_BAUDRATE_230400 = 230400,
  CY_U3P_UART_BAUDRATE_300000 = 300000, CY_U3P_UART_BAUDRATE_400000 = 400000, CY_U3P_UART_BAUDRATE_460800 = 460800, CY_U3P_UART_BAUDRATE_500000 = 500000,
  CY_U3P_UART_BAUDRATE_750000 = 750000, CY_U3P_UART_BAUDRATE_921600 = 921600, CY_U3P_UART_BAUDRATE_1M = 1000000, CY_U3P_UART_BAUDRATE_2M = 2000000,
  CY_U3P_UART_BAUDRATE_3M = 3000000, CY_U3P_UART_BAUDRATE_4M = 4000000, CY_U3P_UART_BAUDRATE_4M608K = 4608000 }

  List of baud rates supported by the UART.
- enum CyU3PUartStopBit_t { CY_U3P_UART_ONE_STOP_BIT = 1, CY_U3P_UART_TWO_STOP_BIT = 2 }

  List of number of stop bits to be used in UART communication.
- enum CyU3PUartParity_t { CY_U3P_UART_NO_PARITY = 0, CY_U3P_UART_EVEN_PARITY, CY_U3P_UART_ODD_PARITY, CY_U3P_UART_NUM_PARITY }

  List of parity settings supported by the UART interface.

Functions

- CyU3PReturnStatus_t CyU3PUartInit (void)

  Starts the UART block on the FX3 device.
- CyU3PReturnStatus_t CyU3PUartDeInit (void)

  Stops the UART hardware block.
- CyU3PReturnStatus_t CyU3PUartSetConfig (CyU3PUartConfig_t *config, CyU3PUartIntrCb_t cb)

  Sets the UART interface parameters.
- CyU3PReturnStatus_t CyU3PUartTxSetBlockXfer (uint32_t txSize)

  Sets the number of bytes to be transmitted by the UART.
- CyU3PReturnStatus_t CyU3PUartRxSetBlockXfer (uint32_t rxSize)

  Sets the number of bytes to be received by the UART.
- uint32_t CyU3PUartTransmitBytes (uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status)

  Transmit data through the UART interface in register mode.
- uint32_t CyU3PUartReceiveBytes (uint8_t *data_p, uint32_t count, CyU3PReturnStatus_t *status)

  Receive data from the UART interface in register mode.
- void CyU3PRegisterUartCallBack (CyU3PUartIntrCb_t uartIntrCb)

  This function registers a callback function for notification of UART interrupts.

5.31.1 Detailed Description

The UART driver in FX3 firmware is responsible for handling data transfers through the UART interface on the device. This file defines the data structures and APIs for UART interface management.

5.31.2 Typedef Documentation

5.31.2.1 typedef enum CyU3PUartBaudrate_t CyU3PUartBaudrate_t

List of baud rates supported by the UART.

Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware. The actual baud rate acheived for SYS_CLK settings of 403.2MHz and 416MHz is also listed below.

See Also

CyU3PUartConfig_t

5.31.2.2 typedef struct CyU3PUartConfig_t CyU3PUartConfig_t

Configuration parameters for the UART interface.

Description

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

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

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

See Also

CyU3PUartBaudrate_t
CyU3PUartStopBit_t
CyU3PUartParity_t
CyU3PUartSetConfig

5.31.2.3 typedef enum CyU3PUartError_t CyU3PUartError_t

List of UART specific error/status codes.

Description

This type lists the various UART specific error/status codes that are sent to the event callback as event data associated with CY_U3P_UART_ERROR_EVT events.

See Also

CyU3PUartEvt_t
CyU3PUartIntrCb_t

5.31.2.4 typedef enum CyU3PUartEvt_t CyU3PUartEvt_t

List of UART related event types.

Description

This enumeration lists the various UART related event codes that are notified to the user application through an event callback.

See Also

CyU3PUartError_t
CyU3PUartIntrCb_t

5.31.2.5 typedef void(* CyU3PUartIntrCb_t)(CyU3PUartEvt_t evt,CyU3PUartError_t error)

Prototype of UART event callback function.

Description

This function type defines a callback to be called when UART interrupts are received. A function of this type can be registered with the UART driver as a callback function and will be called whenever an event of interest occurs.

The UART has to be configured for DMA mode of transfer for callbacks to be registered.

See Also

CyU3PUartEvt_t
CyU3PUartError_t
CyU3PRegisterUartCallBack

5.31.2.6 typedef enum CyU3PUartParity_t CyU3PUartParity_t

List of parity settings supported by the UART interface.

Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

See Also

CyU3PUartConfig_t

5.31.2.7 typedef enum CyU3PUartStopBit_t CyU3PUartStopBit_t

List of number of stop bits to be used in UART communication.

Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have.

See Also

CyU3PUartConfig_t

5.31.3 Enumeration Type Documentation

5.31.3.1 enum CyU3PUartBaudrate_t

List of baud rates supported by the UART.

Description

This enumeration lists the various baud rate settings that are supported by the UART interface and driver implementation. The specific baud rates achieved will be close approximations of these standard values based on the clock frequencies that can be obtained on the FX3 hardware. The actual baud rate acheived for SYS_CLK settings of 403.2MHz and 416MHz is also listed below.

See Also

CyU3PUartConfig_t

Enumerator

CY_U3P_UART_BAUDRATE_100 Baud: 100, Actual @403MHz: 100.00, @416MHz: 100.00,

CY_U3P_UART_BAUDRATE_300 Baud: 300, Actual @403MHz: 300.00, @416MHz: 300.01,

CY_U3P_UART_BAUDRATE_600 Baud: 600, Actual @403MHz: 600.00, @416MHz: 599.99

CY_U3P_UART_BAUDRATE_1200 Baud: 1200, Actual @403MHz: 1200.00, @416MHz: 1200.01

CY_U3P_UART_BAUDRATE_2400 Baud: 2400, Actual @403MHz: 2400.00, @416MHz: 2399.96

CY_U3P_UART_BAUDRATE_4800 Baud: 4800, Actual @403MHz: 4800.00, @416MHz: 4800.15

CY_U3P_UART_BAUDRATE_9600 Baud: 9600, Actual @403MHz: 9600.00, @416MHz: 9599.41

CY_U3P_UART_BAUDRATE_10000 Baud: 10000, Actual @403MHz: 10000.00, @416MHz: 10000.00

CY_U3P_UART_BAUDRATE_14400 Baud: 14400, Actual @403MHz: 14400.00, @416MHz: 14400.44

CY_U3P_UART_BAUDRATE_19200 Baud: 19200, Actual @403MHz: 19200.00, @416MHz: 19202.36

CY_U3P_UART_BAUDRATE_38400 Baud: 38400, Actual @403MHz: 38385.38, @416MHz: 38404.73

CY_U3P_UART_BAUDRATE_50000 Baud: 50000, Actual @403MHz: 50000.00, @416MHz: 50000.00

CY_U3P_UART_BAUDRATE_57600 Baud: 57600, Actual @403MHz: 57600.00, @416MHz: 57585.83

CY_U3P_UART_BAUDRATE_75000 Baud: 75000, Actual @403MHz: 75000.00, @416MHz: 75036.08

CY_U3P_UART_BAUDRATE_100000 Baud: 100000, Actual @403MHz: 100000.00, @416MHz: 100000.00

CY_U3P_UART_BAUDRATE_115200 Baud: 115200, Actual @403MHz: 115068.49, @416MHz: 115299.33

CY_U3P_UART_BAUDRATE_153600 Baud: 153600, Actual @403MHz: 153658.54, @416MHz: 153392.33

CY_U3P_UART_BAUDRATE_200000 Baud: 200000, Actual @403MHz: 200000.00, @416MHz: 200000.00

CY_U3P_UART_BAUDRATE_225000 Baud: 225000, Actual @403MHz: 225000.00, @416MHz: 225108.23

CY_U3P_UART_BAUDRATE_230400 Baud: 230400, Actual @403MHz: 230136.99, @416MHz: 230088.50

CY_U3P_UART_BAUDRATE_300000 Baud: 300000, Actual @403MHz: 300000.00, @416MHz: 300578.03

CY_U3P_UART_BAUDRATE_400000 Baud: 400000, Actual @403MHz: 400000.00, @416MHz: 400000.00

CY_U3P_UART_BAUDRATE_460800 Baud: 460800, Actual @403MHz: 462385.32, @416MHz: 460176.99

CY_U3P_UART_BAUDRATE_500000 Baud: 500000, Actual @403MHz: 499009.90, @416MHz: 500000.00

CY_U3P_UART_BAUDRATE_750000 Baud: 750000, Actual @403MHz: 752238.81, @416MHz: 753623.19

CY_U3P_UART_BAUDRATE_921600 Baud: 921600, Actual @403MHz: 916363.64, @416MHz: 928571.43

CY_U3P_UART_BAUDRATE_1M Baud: 1000000, Actual @403MHz: 1008000.00, @416MHz: 1000000.00

CY_U3P_UART_BAUDRATE_2M Baud: 2000000, Actual @403MHz: 2016000.00, @416MHz: 2000000.00

CY_U3P_UART_BAUDRATE_3M Baud: 3000000, Actual @403MHz: 2964705.88, @416MHz: 3058823.52

CY_U3P_UART_BAUDRATE_4M Baud: 4000000, Actual @403MHz: 3876923.08, @416MHz: 4000000.00

CY_U3P_UART_BAUDRATE_4M608K Baud: 4608000, Actual @403MHz: 4581818.18, @416MHz: 4727272.72

5.31.3.2 enum CyU3PUartError_t

List of UART specific error/status codes.

Description

This type lists the various UART specific error/status codes that are sent to the event callback as event data associated with CY_U3P_UART_ERROR_EVT events.

See Also

CyU3PUartEvt_t
CyU3PUartIntrCb_t

Enumerator

CY_U3P_UART_ERROR_NAK_BYTE_0 Missing stop bit.

CY_U3P_UART_ERROR_RX_PARITY_ERROR RX parity error.

CY_U3P_UART_ERROR_TX_OVERFLOW Overflow of FIFO during transmit operation.

CY_U3P_UART_ERROR_RX_UNDERFLOW Underflow in FIFO during receive/read operation.

CY_U3P_UART_ERROR_RX_OVERFLOW Overflow of FIFO during receive operation.

5.31.3.3 enum CyU3PUartEvt_t

List of UART related event types.

Description

This enumeration lists the various UART related event codes that are notified to the user application through an event callback.

See Also

CyU3PUartError_t
CyU3PUartIntrCb_t

Enumerator

CY_U3P_UART_EVENT_RX_DONE UART data receive operation is complete.

CY_U3P_UART_EVENT_TX_DONE UART data transmit operation is complete.

CY_U3P_UART_EVENT_ERROR An error has been detected.

5.31.3.4 enum CyU3PUartParity_t

List of parity settings supported by the UART interface.

Description

This enumeration lists the various parity settings that the UART interface can be configured to support.

See Also

CyU3PUartConfig_t

Enumerator

CY_U3P_UART_NO_PARITY No parity bits.

CY_U3P_UART_EVEN_PARITY Even parity.

CY_U3P_UART_ODD_PARITY Odd parity.

CY_U3P_UART_NUM_PARITY Number of parity enumerations.

5.31.3.5 enum CyU3PUartStopBit_t

List of number of stop bits to be used in UART communication.

Description

This enumeration lists the various number of stop bit settings that the UART interface can be configured to have.

See Also

CyU3PUartConfig_t

Enumerator

CY_U3P_UART_ONE_STOP_BIT 1 stop bit.

CY_U3P_UART_TWO_STOP_BIT 2 stop bits.

5.31.4 Function Documentation

5.31.4.1 void CyU3PRegisterUartCallBack (CyU3PUartIntrCb_t *uartIntrCb*)

This function registers a callback function for notification of UART interrupts.

Description

This function registers a callback function that will be called for notification of UART interrupts. This can also been done by the CyU3PUartInit API.

Return value

None

See Also

CyU3PUartIntrCb_t
CyU3PUartInit

Parameters

uartIntrCb	Callback function pointer.

5.31.4.2 CyU3PReturnStatus_t CyU3PUartDeInit (void)

Stops the UART hardware block.

Description

This function disables and powers off the UART hardware block on the device.

Return value

CY_U3P_SUCCESS - if the de-init was successful.

CY_U3P_ERROR_NOT_STARTED - if UART had not been initialized.

See Also

CyU3PUartInit

5.31.4.3 CyU3PReturnStatus_t CyU3PUartInit (void)

Starts the UART block on the FX3 device.

Description

This function powers up the UART hardware block on the device and should be the first UART related function called by the application.

Return value

CY_U3P_SUCCESS - if the init was successful.

CY_U3P_ERROR_ALREADY_STARTED - if the UART block had been previously initialized.

CY_U3P_ERROR_NOT_CONFIGURED - if UART was not enabled during IO configuration.

See Also

CyU3PUartDeInit
CyU3PUartSetConfig
CyU3PUartTransmitBytes
CyU3PUartReceiveBytes
CyU3PUartTxSetBlockXfer
CyU3PUartRxSetBlockXfer

5.31.4.4 uint32_t CyU3PUartReceiveBytes (uint8_t * *data_p*, uint32_t *count*, CyU3PReturnStatus_t * *status*)

Receive data from the UART interface in register mode.

Description

This function is used to read the specified number of bytes from the UART in register mode. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

Return value

Number of bytes that are successfully received.

See Also

CyU3PUartSetConfig
CyU3PUartTransmitBytes

Parameters

data_p	Pointer to location where the data read is to be placed.
count	Number of bytes to be received.
status	Status returned from the operation.

5.31.4.5 CyU3PReturnStatus_t CyU3PUartRxSetBlockXfer (uint32_t *rxSize*)

Sets the number of bytes to be received by the UART.

Description

This function sets the size of the desired data reception through the UART. The value 0xFFFFFFFFU can be used to specify infinite or indefinite data reception.

This function is to be used when the UART is configured for DMA mode of transfer. If this function is called when the UART is configured in register mode, it will return with an error.

Return value

CY_U3P_SUCCESS - if the transfer size was set successfully.

CY_U3P_ERROR_NOT_CONFIGURED - if UART was not configured for DMA mode.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the UART block.

See Also

CyU3PUartSetConfig
CyU3PUartTxSetBlockXfer

Parameters

rxSize	Desired transfer size.

5.31.4.6 CyU3PReturnStatus_t CyU3PUartSetConfig (CyU3PUartConfig_t * *config*, CyU3PUartIntrCb_t *cb*)

Sets the UART interface parameters.

Description

This function configures the UART block with the desired user parameters such as transfer mode, baud rate etc. This function should be called repeatedly to make any change to the set of configuration parameters.

Baudrate calculation: The FX3 UART supports baud rates between 100 Hz and 4 MHz. The UART block needs to be clocked at 16X the external baud rate. Since the block clock is derived from the SYS_CLK frequency through integer and half dividers, it is only possible to set the baud rate to an approximation of the desired value.

See the documentation for the CyU3PUartBaudrate_t for a list of actual baud rates that are achieved for various settings.

Return value

CY_U3P_SUCCESS - if the configuration was set successfully.

CY_U3P_ERROR_NOT_STARTED - if UART was not initialized.

CY_U3P_ERROR_NULL_POINTER - if a NULL pointer is passed.

CY_U3P_ERROR_BAD_ARGUMENT - if any of the parameters are invalid.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the UART block.

See Also

CyU3PUartConfig_t
CyU3PUartIntrCb_t
CyU3PUartInit
CyU3PUartTransmitBytes
CyU3PUartReceiveBytes
CyU3PUartTxSetBlockXfer
CyU3PUartRxSetBlockXfer

Parameters

config	Pointer to structure containing config information.
cb	UART callback function to be registered.

5.31.4.7 uint32_t CyU3PUartTransmitBytes (uint8_t * *data_p*, uint32_t *count*, CyU3PReturnStatus_t * *status*)

Transmit data through the UART interface in register mode.

Description

This function is used to transfer the specified number of bytes out through the UART in register mode. This function can only be used if the UART has been configured for register (non-DMA) transfer mode.

Return value

Number of bytes that are successfully transferred.

See Also

CyU3PUartSetConfig
CyU3PUartReceiveBytes

Parameters

data_p	Pointer to the data to be transferred.
count	Number of bytes to be transferred.
status	Status returned from the operation.

5.31.4.8 CyU3PReturnStatus_t CyU3PUartTxSetBlockXfer (uint32_t *txSize*)

Sets the number of bytes to be transmitted by the UART.

Description

This function sets the size of the desired data transmission through the UART. The value 0xFFFFFFFFU can be used to specify infinite or indefinite data transmission.

This function is to be used when the UART is configured for DMA mode of transfer. If this function is called when the UART is configured in register mode, it will return with an error.

Return value

CY_U3P_SUCCESS - if the transfer size was set successfully.

CY_U3P_ERROR_NOT_CONFIGURED - if UART was not configured for DMA mode.

CY_U3P_ERROR_MUTEX_FAILURE - Failed to get a mutex lock on the UART block.

See Also

CyU3PUartSetConfig
CyU3PUartRxSetBlockXfer

Parameters

txSize	Desired transfer size.

5.32 firmware/u3p_firmware/inc/cyu3usb.h File Reference

The USB device mode driver on the FX3 device is responsible for handling USB 3.0 and 2.0 connections, and providing the API hooks for configuring device operations.

```
#include "cyu3usbconst.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Data Structures

- struct CyU3PEpConfig_t

  Endpoint configuration information.

Macros

- #define CY_U3P_USB_REQUEST_TYPE_MASK (0x000000FF)

  Mask to get the bmRequestType field from the USB setup request.
- #define CY_U3P_USB_REQUEST_TYPE_POS (0)

  Position of the bmRequestType field in the USB setup request.
- #define CY_U3P_USB_REQUEST_MASK (0x0000FF00)

  Mask to get the bRequest field from the USB setup request.
- #define CY_U3P_USB_REQUEST_POS (8)

  Position of the bRequest field in the USB setup request.
- #define CY_U3P_USB_VALUE_MASK (0xFFFF0000)

  Mask to get the wValue field from the USB setup request.
- #define CY_U3P_USB_VALUE_POS (16)

  Position of the wValue field in the USB setup request.
- #define CY_U3P_USB_INDEX_MASK (0x0000FFFF)

  Mask to get the wIndex field from the USB setup request.
- #define CY_U3P_USB_INDEX_POS (0)

  Position of the wIndex field in the USB setup request.
- #define CY_U3P_USB_LENGTH_MASK (0xFFFF0000)

  Mask to get the wLength field from the USB setup request.
- #define CY_U3P_USB_LENGTH_POS (16)

  Position of the wLength field in the USB setup request.
- #define CY_U3P_MAX_STRING_DESC_INDEX (16)

  Number of string descriptors that are supported by the USB driver.
- #define **CYU3P_USB_LOG_VBUS_OFF** (0x01) /* Indicates VBus turned off. */
- #define **CYU3P_USB_LOG_VBUS_ON** (0x02) /* Indicates VBus turned on. */
- #define **CYU3P_USB_LOG_USB2_PHY_OFF** (0x03) /* Indicates that the 2.0 PHY has been turned off. */
- #define **CYU3P_USB_LOG_USB3_PHY_OFF** (0x04) /* Indicates that the 3.0 PHY has been turned off. */
- #define **CYU3P_USB_LOG_USB2_PHY_ON** (0x05) /* Indicates that the 2.0 PHY has been turned on. */
- #define **CYU3P_USB_LOG_USB3_PHY_ON** (0x06) /* Indicates that the 3.0 PHY has been turned on. */
- #define **CYU3P_USB_LOG_USBSS_DISCONNECT** (0x10) /* Indicates that the USB 3.0 link has been disabled. */
- #define **CYU3P_USB_LOG_USBSS_RESET** (0x11) /* Indicates that a USB 3.0 reset (warm/hot) has happened. */
- #define **CYU3P_USB_LOG_USBSS_CONNECT** (0x12) /* Indicates that USB 3.0 Rx Termination has been detected. */
- #define **CYU3P_USB_LOG_USBSS_CTRL** (0x14) /* Indicates that a USB 3.0 control request has been received. */
- #define **CYU3P_USB_LOG_USBSS_STATUS** (0x15) /* Indicates completion of status stage for a 3.0 control request. */
- #define **CYU3P_USB_LOG_USBSS_ACKSETUP** (0x16) /* Indicates that the CyU3PUsbAckSetup API has been called. */
- #define **CYU3P_USB_LOG_LGO_U1** (0x21) /* Indicates that a LGO_U1 command has been received. */
- #define **CYU3P_USB_LOG_LGO_U2** (0x22) /* Indicates that a LGO_U2 command has been received. */
- #define **CYU3P_USB_LOG_LGO_U3** (0x23) /* Indicates that a LGO_U3 command has been received. */
- #define **CYU3P_USB_LOG_USB2_SUSP** (0x40) /* Indicates that a USB 2.0 suspend condition has been detected. */
- #define **CYU3P_USB_LOG_USB2_RESET** (0x41) /* Indicates that a USB 2.0 bus reset has been detected. */
- #define **CYU3P_USB_LOG_USB2_HSGRANT** (0x42) /* Indicates that the USB High-Speed handshake has been completed. */
- #define **CYU3P_USB_LOG_USB2_CTRL** (0x44) /* Indicates that a FS/HS control request has been received. */

- #define **CYU3P_USB_LOG_USB2_STATUS** (0x45) /∗ Indicates completion of status stage for a FS/HS control transfer. ∗/
- #define **CYU3P_USB_LOG_USB_FALLBACK** (0x50) /∗ Indicates that the USB connection is dropping from 3.0 to 2.0 ∗/
- #define **CYU3P_USB_LOG_USBSS_ENABLE** (0x51) /∗ Indicates that a USB 3.0 connection is being attempted again. ∗/
- #define **CYU3P_USB_LOG_USBSS_LNKERR** (0x52) /∗ The number of link errors has crossed the threshold. ∗/
- #define **CYU3P_USB_LOG_LTSSM_CHG** (0x80) /∗ Base of values that indicate a USB 3.0 LTSSM state change. ∗/

Typedefs

- typedef enum CyU3PUsbEventType_t CyU3PUsbEventType_t

  List of USB related events that are raised by the USB manager.
- typedef enum CyU3PUsbMgrStates_t CyU3PUsbMgrStates_t

  List of USB device manager states.
- typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t

  Virtual descriptor types to be used to set descriptors.
- typedef enum CyU3PUSBSpeed_t CyU3PUSBSpeed_t

  List of supported USB connection speeds.
- typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode

  List of USB 3.0 link operating modes.
- typedef enum CyU3PUsbEpEvtType CyU3PUsbEpEvtType

  List of USB endpoint events.
- typedef void(∗ CyU3PUsbEpEvtCb_t)(CyU3PUsbEpEvtType evType, CyU3PUSBSpeed_t usbSpeed, uint8_t epNum)

  Callback function type used for notification of USB endpoint events.
- typedef void(∗ CyU3PUSBEventCb_t)(CyU3PUsbEventType_t evtype, uint16_t evdata)

  USB event callback type.
- typedef CyBool_t(∗ CyU3PUSBSetupCb_t)(uint32_t setupdat0, uint32_t setupdat1)

  USB setup request handler type.
- typedef CyBool_t(∗ CyU3PUsbLPMReqCb_t)(CyU3PUsbLinkPowerMode link_mode)

  Event callback raised on host request to switch USB link to a low power state.
- typedef struct CyU3PEpConfig_t CyU3PEpConfig_t

  Endpoint configuration information.
- typedef enum CyU3PUsbDevProperty CyU3PUsbDevProperty

  List of USB device properties that can be queried.

Enumerations

- enum CyU3PUsbEventType_t {
  CY_U3P_USB_EVENT_CONNECT = 0, CY_U3P_USB_EVENT_DISCONNECT, CY_U3P_USB_EVENT_SUSPEND, CY_U3P_USB_EVENT_RESUME,
  CY_U3P_USB_EVENT_RESET, CY_U3P_USB_EVENT_SETCONF, CY_U3P_USB_EVENT_SPEED, CY_U3P_USB_EVENT_SETINTF,
  CY_U3P_USB_EVENT_SET_SEL, CY_U3P_USB_EVENT_SOF_ITP, CY_U3P_USB_EVENT_EP0_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,
  CY_U3P_USB_EVENT_USB3_LNKFAIL, CY_U3P_USB_EVENT_SS_COMP_ENTRY, CY_U3P_USB_EVENT_SS_COMP_EXIT }

List of USB related events that are raised by the USB manager.

- enum CyU3PUsbMgrStates_t { CY_U3P_USB_INACTIVE = 0, CY_U3P_USB_STARTED, CY_U3P_USB_WAITING_FOR_DESC, CY_U3P_USB_CONFIGURED, CY_U3P_USB_VBUS_WAIT, CY_U3P_USB_CONNECTED, CY_U3P_USB_ESTABLISHED, CY_U3P_USB_MS_ACTIVE, CY_U3P_USB_MAX_STATE }

  List of USB device manager states.
- enum CyU3PUSBSetDescType_t { CY_U3P_USB_SET_SS_DEVICE_DESCR, CY_U3P_USB_SET_HS_DEVICE_DESCR, CY_U3P_USB_SET_DEVQUAL_DESCR, CY_U3P_USB_SET_FS_CONFIG_DESCR, CY_U3P_USB_SET_HS_CONFIG_DESCR, CY_U3P_USB_SET_STRING_DESCR, CY_U3P_USB_SET_SS_CONFIG_DESCR, CY_U3P_USB_SET_SS_BOS_DESCR, CY_U3P_USB_SET_SS_DEVICE_DESCR, CY_U3P_USB_SET_HS_DEVICE_DESCR, CY_U3P_USB_SET_DEVQUAL_DESCR, CY_U3P_USB_SET_FS_CONFIG_DESCR, CY_U3P_USB_SET_HS_CONFIG_DESCR, CY_U3P_USB_SET_STRING_DESCR, CY_U3P_USB_SET_SS_CONFIG_DESCR, CY_U3P_USB_SET_SS_BOS_DESCR, CY_U3P_USB_SET_OTG_DESCR }

  Virtual descriptor types to be used to set descriptors.
- enum CyU3PUSBSpeed_t { CY_U3P_NOT_CONNECTED = 0x00, CY_U3P_FULL_SPEED, CY_U3P_HIGH_SPEED, CY_U3P_SUPER_SPEED }

  List of supported USB connection speeds.
- enum CyU3PUsbLinkPowerMode { CyU3PUsbLPM_U0 = 0, CyU3PUsbLPM_U1, CyU3PUsbLPM_U2, CyU3PUsbLPM_U3, CyU3PUsbLPM_COMP, CyU3PUsbLPM_Unknown }

  List of USB 3.0 link operating modes.
- 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 }

  List of USB endpoint events.
- 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 }

  List of USB device properties that can be queried.

Functions

- void CyU3PUsbRegisterEventCallback (CyU3PUSBEventCb_t callback)

  Register a USB event callback function.
- void CyU3PUsbRegisterSetupCallback (CyU3PUSBSetupCb_t callback, CyBool_t fastEnum)

  Register a USB setup request handler.
- void CyU3PUsbRegisterLPMRequestCallback (CyU3PUsbLPMReqCb_t cb)

  Register a USB 3.0 LPM request handler callback.
- void CyU3PUsbRegisterEpEvtCallback (CyU3PUsbEpEvtCb_t cbFunc, uint32_t eventMask, uint16_t outEpMask, uint16_t inEpMask)

  Register a callback function for notification of USB endpoint events.
- CyBool_t CyU3PUsbIsStarted (void)

  This function returns whether the USB device module has been started.
- CyU3PReturnStatus_t CyU3PUsbStart (void)

  Start the USB driver.
- CyU3PReturnStatus_t CyU3PUsbStop (void)

Stop the USB driver.

- CyU3PReturnStatus_t CyU3PUsbSetDesc (CyU3PUSBSetDescType_t desc_type, uint8_t desc_index, uint8_t *desc)

  Register a USB descriptor with the driver.
- CyU3PReturnStatus_t CyU3PConnectState (CyBool_t connect, CyBool_t ssEnable)

  Enable / disable the USB connection.
- CyU3PReturnStatus_t CyU3PUsbStall (uint8_t ep, CyBool_t stall, CyBool_t toggle)

  Set or clear the stall status of an endpoint.
- CyU3PReturnStatus_t CyU3PUsbGetEpCfg (uint8_t ep, CyBool_t *isNak, CyBool_t *isStall)

  Retrieve the current state of the specified endpoint.
- CyU3PReturnStatus_t CyU3PUsbSetEpNak (uint8_t ep, CyBool_t nak)

  Force or clear the NAK status on the specified endpoint.
- CyBool_t CyU3PGetConnectState (void)

  Check whether the FX3 device is currently connected to a host.
- void CyU3PUsbAckSetup (void)

  Complete the status handshake of a USB control request.
- CyU3PReturnStatus_t CyU3PSetEpConfig (uint8_t ep, CyU3PEpConfig_t *epinfo)

  Configure a USB endpoint.
- CyU3PReturnStatus_t CyU3PSetEpPacketSize (uint8_t ep, uint16_t maxPktSize)

  Set the maximum packet size for a USB endpoint.
- CyU3PReturnStatus_t CyU3PUsbSetEpPktMode (uint8_t ep, CyBool_t pktMode)

  Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.
- CyU3PReturnStatus_t CyU3PUsbSendEP0Data (uint16_t count, uint8_t *buffer)

  Send data to the USB host via EP0-IN.
- CyU3PReturnStatus_t CyU3PUsbGetEP0Data (uint16_t count, uint8_t *buffer, uint16_t *readCount)

  Read data associated with a control-OUT transfer.
- CyU3PReturnStatus_t CyU3PUsbFlushEp (uint8_t ep)

  Clear (flush) all data buffers associated with a specified endpoint.
- CyU3PReturnStatus_t CyU3PUsbResetEp (uint8_t ep)

  Resets the specified endpoint.
- CyU3PReturnStatus_t CyU3PUsbMapStream (uint8_t ep, uint8_t socketNum, uint16_t streamId)

  Map a socket to stream of the specified endpoint.
- CyU3PReturnStatus_t CyU3PUsbChangeMapping (uint8_t ep, uint8_t socketNum, CyBool_t remap, uint16_t newstreamId, uint8_t newep)

  Map a socket to stream of the specified endpoint.
- CyU3PUSBSpeed_t CyU3PUsbGetSpeed (void)

  Get the connection speed at which USB is operating.
- CyU3PReturnStatus_t CyU3PUsbVBattEnable (CyBool_t enable)

  Configure USB block on FX3 to work off Vbatt power instead of Vbus.
- CyU3PReturnStatus_t CyU3PUsbGetDevProperty (CyU3PUsbDevProperty type, uint32_t *buf)

  Function that returns some properties of the USB device.
- CyU3PReturnStatus_t CyU3PUsbSendErdy (uint8_t ep, uint16_t bulkStream)

  Function to send an ERDY TP to a USB 3.0 host.
- CyU3PReturnStatus_t CyU3PUsbSendNrdy (uint8_t ep, uint16_t bulkStream)

  Function to send an NRDY TP to a USB 3.0 host.
- CyU3PReturnStatus_t CyU3PUsbGetLinkPowerState (CyU3PUsbLinkPowerMode *mode_p)

  Function to get the current power state of the USB 3.0 link.
- CyU3PReturnStatus_t CyU3PUsbSetLinkPowerState (CyU3PUsbLinkPowerMode link_mode)

  Function to request a device entry into one of the U0, U1 or U2 link power modes.
- CyU3PReturnStatus_t CyU3PUsbEnableITPEvent (CyBool_t enable)

  Function to enable/disable notification of SOF/ITP events to the application.

- CyU3PReturnStatus_t CyU3PUsbDoRemoteWakeup (void)

  Trigger USB 2.0 Remote Wakeup signalling to the host.
- CyU3PReturnStatus_t CyU3PUsbForceFullSpeed (CyBool_t enable)

  Function to force the USB 2.0 device connection to full speed.
- CyU3PReturnStatus_t CyU3PUsbSendDevNotification (uint8_t notificationType, uint32_t param0, uint32_t param1)

  Function to send a DEV_NOTIFICATION Transaction Packet to the host.
- CyU3PReturnStatus_t CyU3PUsbLPMDisable (void)

  Function to prevent FX3 from entering U1/U2 states.
- CyU3PReturnStatus_t CyU3PUsbLPMEnable (void)

  Function to re-enable automated handling of U1/U2 state entry.
- void CyU3PUsbSetBooterSwitch (CyBool_t enable)

  Function to enable/disable switching control back to the FX3 2-stage bootloader.
- CyU3PReturnStatus_t CyU3PUsbGetBooterVersion (uint8_t *major_p, uint8_t *minor_p, uint8_t *patch_p)

  Function to check the version of the boot firmware that transferred control to this firmware image.
- CyU3PReturnStatus_t CyU3PUsbJumpBackToBooter (uint32_t address)

  Function to transfer control back to the FX3 2-stage bootloader.
- void CyU3PUsbInitEventLog (uint8_t *buffer, uint32_t bufSize)

  Function to initiate logging of USB state changes into a circular buffer.
- uint16_t CyU3PUsbGetEventLogIndex (void)

  Get the current event log buffer index.
- void CyU3PUsbEpPrepare (CyU3PUSBSpeed_t curSpeed)

  Prepare all enabled USB device endpoints for data transfer.
- CyU3PReturnStatus_t CyU3PUsbResetEndpointMemories (void)

  Reset and re-initialize the endpoint memory block on FX3.
- void CyU3PUsbEnableEPPrefetch (void)

  Configure the USB DMA interface to continuously pre-fetch data.
- CyU3PReturnStatus_t CyU3PUsbGetErrorCounts (uint16_t *phy_err_cnt, uint16_t *lnk_err_cnt)

  Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.
- CyU3PReturnStatus_t CyU3PUsbEPSetBurstMode (uint8_t ep, CyBool_t burstEnable)

  Function to enable burst mode operation for a USB endpoint.
- CyU3PReturnStatus_t CyU3PUsbSetTxSwing (uint32_t swing)

  Set the Tx amplitude range for the USB 3.0 signals.
- CyU3PReturnStatus_t CyU3PUsbSetTxDeemphasis (uint32_t value)

  Set the Tx de-emphasis setting for the USB 3.0 signals.
- CyU3PReturnStatus_t CyU3PUsbGetEpSeqNum (uint8_t ep, uint8_t *seqnum_p)

  Get the current sequence number for an endpoint.
- CyU3PReturnStatus_t CyU3PUsbSetEpSeqNum (uint8_t ep, uint8_t seqnum)

  Set the active sequence number for an endpoint.
- CyU3PReturnStatus_t CyU3PUsbControlVBusDetect (CyBool_t enable, CyBool_t useVbatt)

  Enable/Disable VBus detection in FX3 firmware.
- CyU3PReturnStatus_t CyU3PUsbControlUsb2Support (CyBool_t enable)

  Enable/Disable USB 2.0 device operation on FX3 device.

5.32.1 Detailed Description

The USB device mode driver on the FX3 device is responsible for handling USB 3.0 and 2.0 connections, and providing the API hooks for configuring device operations.

5.32.2 Enumeration Modes

The FX3 USB driver supports two different modes of enumeration, one of which is optimized for good performance and simplified application code; whereas the other allows for greater flexibility.

Description

Two different modes of enumeration control are supported by the USB driver on the FX3 device.

1. Fast enumeration: In this case, all of the descriptors for all USB speeds of interest are registered with the USB driver through the CyU3PUsbSetDesc API call. The USB driver then uses this information to handle all of the standard USB setup requests. The driver is responsible for identifying the current connection speed and using the appropriate set of USB descriptors.

In this case, only a single configuration descriptor can be used because only one config descriptor for each connection speed can be registered with the USB driver. The application will also need to disconnect the USB link and re-enumerate every time it wants to change the descriptors.

2. User enumeration: In this case, all USB setup requests are forwarded to the user application logic through a callback function. It is the responsibility of the application code to handle these requests and to send the appropriate responses to the USB host. The application also has to track the current USB connection speed and use this to identify the specific descriptor to send up to the host.

In this case, the application can change the descriptor data at will and has maximum flexibility in terms of implementing multiple configurations and alternate settings.

Note

In either form of enumeration, any non-standard (Class or vendor specific) setup request will trigger a callback function. The user application is expected to handle these requests in all cases.

5.32.3 USB Device Mode Functions

The USB device APIs are used to configure the USB device functionality and to perform USB data transfers.

Description

The FX3 device supports programmable USB peripheral implementation at USB-SS, USB-HS and USB-FS speeds. The type of USB device to be implemented as well as the endpoint pipes to be used can be configured through a set of USB device mode APIs. The USB device mode APIs also provide the capability to manage the endpoint states such as STALL and NAK as well as to perform data transfers on the control endpoint (EP0). Data transfers on the other USB endpoints will be controlled through the DMA APIs.

5.32.4 Macro Definition Documentation

5.32.4.1 #define CY_U3P_MAX_STRING_DESC_INDEX (16)

Number of string descriptors that are supported by the USB driver.

Description

The USB driver is capable of storing pointers to and handling a number of string descriptors. This constant represents the number of strings that the driver is capable of handling.

See Also

CyU3PUsbSetDesc

5.32.5 Typedef Documentation

5.32.5.1 typedef struct CyU3PEpConfig_t CyU3PEpConfig_t

Endpoint configuration information.

Description

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

See Also

CyU3PSetEpConfig

5.32.5.2 typedef enum CyU3PUsbDevProperty CyU3PUsbDevProperty

List of USB device properties that can be queried.

Description

This is the list of USB device properties that can be queried by the application through the CyU3PUsbGetDevProperty API. Each property value is specified as a 32 bit integer value.

See Also

CyU3PUsbGetDevProperty

5.32.5.3 typedef void(∗ CyU3PUsbEpEvtCb_t)(CyU3PUsbEpEvtType evType,CyU3PUSBSpeed_t usbSpeed,uint8_t epNum)

Callback function type used for notification of USB endpoint events.

Description

This callback function type is used by the firmware application to receive notifications of USB endpoint events of interest.

See Also

CyU3PUsbEpEvtType
CyU3PUsbRegisterEpEvtCallback

5.32.5.4 typedef enum CyU3PUsbEpEvtType CyU3PUsbEpEvtType

List of USB endpoint events.

Description

The USB API can relay a set of endpoint related events to the firmware application for monitoring and control purposes. This enumerated data type lists the various endpoint events that can be generated by the API library.

See Also

CyU3PUsbEpEvtCb_t
CyU3PUsbRegisterEpEvtCallback

5.32.5.5 typedef void(* CyU3PUSBEventCb_t)(CyU3PUsbEventType_t evtype,uint16_t evdata)

USB event callback type.

Description

Type of callback function that should be registered with the USB driver to get notifications of USB specific events.

The evtype parameter specifies the type of event and evdata contains event specific data.

See Also

CyU3PUsbEventType_t
CyU3PUsbRegisterEventCallback

5.32.5.6 typedef enum CyU3PUsbEventType_t CyU3PUsbEventType_t

List of USB related events that are raised by the USB manager.

Description

The USB manager or driver continually runs on a thread on the FX3 device and performs the necessary operations to handle various USB requests from the host. The USB driver can notify the user application about various USB specific events by calling a callback function that is registered with it. This type lists the various USB event types that are notified to the application logic.

See Also

CyU3PUSBEventCb_t
CyU3PUsbEnableITPEvent

5.32.5.7 typedef enum CyU3PUsbLinkPowerMode CyU3PUsbLinkPowerMode

List of USB 3.0 link operating modes.

Description

List of USB 3.0 link operating modes. Only the active and low power modes are listed here as the rest of the link modes are not visible to software.

See Also

CyU3PUsbLPMReqCb_t

5.32.5.8 typedef CyBool_t(* CyU3PUsbLPMReqCb_t)(CyU3PUsbLinkPowerMode link_mode)

Event callback raised on host request to switch USB link to a low power state.

Description

The USB 3.0 host may request the FX3 device to switch the USB link to a low power state when the host has no more activity to perform. This event callback is raised when a request from the host to switch the link to a U1/U2 state is received. The link_mode parameter indicates the power state that the host is requesting for. The return value from this function call is expected to indicate whether the low power state entry request should be accepted. A return of CyTrue will cause FX3 to accept entry into U1/U2 state and a return of CyFalse will cause FX3 to reject the request.

See Also

CyU3PUsbLinkPowerMode
CyU3PUsbRegisterLPMRequestCallback

5.32.5.9 typedef enum CyU3PUsbMgrStates_t CyU3PUsbMgrStates_t

List of USB device manager states.

Description

This USB driver (manager) implements a state machine that tracks the current state of the USB connection and uses this to identify the appropriate response to USB requests. This type lists the various possible states for the USB driver state machine. Some of these states are software defined and have no connection with any USB bus conditions.

5.32.5.10 typedef enum CyU3PUSBSetDescType_t CyU3PUSBSetDescType_t

Virtual descriptor types to be used to set descriptors.

Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

See Also

CyU3PUsbSetDesc

5.32.5.11 typedef CyBool_t(∗ CyU3PUSBSetupCb_t)(uint32_t setupdat0,uint32_t setupdat1)

USB setup request handler type.

Description

Type of callback function that is invoked to handle USB setup requests. The function shall return CyTrue (1) if it has handled the setup request. If it is unable to handle this specific request, it shall return CyFalse and the USB driver will perform the default actions corresponding to the setup request.

The handling of each setup request will involve one and only one of the following API calls:

CyU3PUsbSendEP0Data: Called to perform any IN-data phase associated with the request. The status handshake for the request will be completed as soon as all of the data requested has been sent to the host.

CyU3PUsbGetEP0Data: Called to perform any OUT-data phase associated with the request. The length specified should match the transfer length specified by the host. The status handshake for the request will be completed as soon as all of the data has been received from the host.

CyU3PUsbAckSetup: Called to complete the status handshake (ACK) phase of a request that does not involve any data transfer.

CyU3PUsbStall: Called to stall EP0 to fail the setup request. The ep number can be specified as 0x00 or 0x80, and the API will take care of stalling the ep in the appropriate direction.

These API calls can be made from the Setup callback directly or in a delayed fashion in a bottom half that is scheduled from the Setup callback. In either case, the Setup callback should return CyTrue to let the API know that the default action (stalling EP0 for any non-standard request) should not be performed.

Note

Calling both CyU3PUsbSendEP0Data/CyU3PUsbGetEP0Data and CyU3PUsbAckSetup or calling CyU3PUsbAckSetup multiple times in response to a control request can result in errors due to the subsequent request being ACKed prematurely.

See Also

CyU3PUsbRegisterSetupCallback
CyU3PUsbSendEP0Data
CyU3PUsbGetEP0Data
CyU3PUsbAckSetup
CyU3PUsbStall

5.32.6 Enumeration Type Documentation

5.32.6.1 enum CyU3PUsbDevProperty

List of USB device properties that can be queried.

Description

This is the list of USB device properties that can be queried by the application through the CyU3PUsbGetDevProperty API. Each property value is specified as a 32 bit integer value.

See Also

CyU3PUsbGetDevProperty

Enumerator

CY_U3P_USB_PROP_DEVADDR The USB device address assigned to FX3. The 7 bit device address value is returned as the LS bits of a 32 bit word.

CY_U3P_USB_PROP_FRAMECNT USB 2.0 frame count value. Returned as a 32 bit unsigned integer value.

CY_U3P_USB_PROP_LINKSTATE Current state of the USB 3.0 Link. Returned as a 32 bit unsigned integer value. See CyU3PUsbLinkState_t for state encoding values.

CY_U3P_USB_PROP_ITPINFO The Isochronous timestamp field from the last ITP received. Returned as a 32 bit unsigned integer value.

CY_U3P_USB_PROP_SYS_EXIT_LAT The System Exit Latency value specified by the USB host. The six byte SEL data is copied into the data buffer. The pointer passed in this case should correspond to a six byte (or longer) buffer.

5.32.6.2 enum CyU3PUsbEpEvtType

List of USB endpoint events.

Description

The USB API can relay a set of endpoint related events to the firmware application for monitoring and control purposes. This enumerated data type lists the various endpoint events that can be generated by the API library.

See Also

CyU3PUsbEpEvtCb_t
CyU3PUsbRegisterEpEvtCallback

Enumerator

CYU3P_USBEP_NAK_EVT The endpoint sent NAK in response to a host transfer request. In the case of a USB 3.0 connection, this callback only indicates that the endpoint has gone into a flowcontrol state due to not being ready for data transfer. This does not necessarily mean that the device has sent an NRDY response to the USB host on that endpoint.

CYU3P_USBEP_ZLP_EVT A Zero Length Packet was sent/received on the endpoint.

CYU3P_USBEP_SLP_EVT A Short Length Packet was sent/received on the endpoint.

CYU3P_USBEP_ISOERR_EVT Out of sequence ISO PIDs or ZLP received on ISO endpoint.

CYU3P_USBEP_SS_RETRY_EVT An ACK TP with the RETRY bit was received on the IN endpoint.

CYU3P_USBEP_SS_SEQERR_EVT Sequence number error detected during endpoint data transfer.

CYU3P_USBEP_SS_STREAMERR_EVT A Bulk-Stream protocol error occured on the endpoint.

5.32.6.3 enum CyU3PUsbEventType_t

List of USB related events that are raised by the USB manager.

Description

The USB manager or driver continually runs on a thread on the FX3 device and performs the necessary operations to handle various USB requests from the host. The USB driver can notify the user application about various USB specific events by calling a callback function that is registered with it. This type lists the various USB event types that are notified to the application logic.

See Also

CyU3PUSBEventCb_t
CyU3PUsbEnableITPEvent

Enumerator

CY_U3P_USB_EVENT_CONNECT USB Connect event. The evData parameter is an integer value which indicates whether it is a 3.0 connection (evData = 1) or a 2.0 connection (evData = 0).

CY_U3P_USB_EVENT_DISCONNECT USB Disconnect event. The evData parameter is not used and will be NULL.

CY_U3P_USB_EVENT_SUSPEND USB Suspend event for both USB 2.0 and 3.0 connections. The evData parameter is not used and will be NULL.

CY_U3P_USB_EVENT_RESUME USB Resume event. The evData parameter is not used and will be NULL.

CY_U3P_USB_EVENT_RESET USB Reset event. The evData parameter is an integer value which indicates whether it is a 3.0 connection (evData = 1) or a 2.0 connection (evData = 0).

CY_U3P_USB_EVENT_SETCONF USB Set Configuration event. evData provides the configuration number that is selected by the host.

CY_U3P_USB_EVENT_SPEED USB Speed Change event. The evData parameter is not used and will always be set to 1.

CY_U3P_USB_EVENT_SETINTF USB Set Interface event. The evData parameter provides the interface number and the selected alternate setting values. Bits 7 - 0: LSB of wValue field from setup request. Bits 15 - 8: LSB of wIndex field from setup request.

CY_U3P_USB_EVENT_SET_SEL USB 3.0 Set System Exit Latency event. The event data parameter will be zero. The CyU3PUsbGetDevProperty API can be used to get the SEL values specified by the host.

CY_U3P_USB_EVENT_SOF_ITP SOF/ITP event notification. The evData parameter is not used and will be NULL. Since these events are very frequent, the event notification needs to be separately enabled using the CyU3PUsbEnableITPEvent() API call.

CY_U3P_USB_EVENT_EP0_STAT_CPLT Event notifying completion of the status phase of a control request. This event is only generated for control requests that are handled by the user's application; and not for requests that are handled internally by the USB driver.

CY_U3P_USB_EVENT_VBUS_VALID Vbus power detected on FX3's USB port.

CY_U3P_USB_EVENT_VBUS_REMOVED Vbus power removed from FX3's USB port.

CY_U3P_USB_EVENT_HOST_CONNECT USB host mode connect event.

CY_U3P_USB_EVENT_HOST_DISCONNECT USB host mode disconnect event.

CY_U3P_USB_EVENT_OTG_CHANGE USB OTG-ID Pin state change.

CY_U3P_USB_EVENT_OTG_VBUS_CHG VBus made valid by OTG host.

CY_U3P_USB_EVENT_OTG_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.

CY_U3P_USB_EVENT_USB3_LNKFAIL Indicates that the USB 3.0 link from FX3 to the USB host has failed. This event is only generated if the USB 2.0 operation of FX3 is disabled; and should be used to trigger a USB 2.0 connection from the external controller.

CY_U3P_USB_EVENT_SS_COMP_ENTRY Indicates that the USB 3.0 link has entered the compliance state.

CY_U3P_USB_EVENT_SS_COMP_EXIT Indicates that the USB 3.0 link has exited the compliance state.

5.32.6.4 enum CyU3PUsbLinkPowerMode

List of USB 3.0 link operating modes.

Description

List of USB 3.0 link operating modes. Only the active and low power modes are listed here as the rest of the link modes are not visible to software.

See Also

CyU3PUsbLPMReqCb_t

Enumerator

CyU3PUsbLPM_U0 U0 (active) power state.

CyU3PUsbLPM_U1 U1 power state.

CyU3PUsbLPM_U2 U2 power state.

CyU3PUsbLPM_U3 U3 (suspend) power state.

CyU3PUsbLPM_COMP Compliance state.

CyU3PUsbLPM_Unknown The link is not in any of the Ux states. This normally happens while the link training is ongoing.

5.32.6.5 enum CyU3PUsbMgrStates_t

List of USB device manager states.

Description

This USB driver (manager) implements a state machine that tracks the current state of the USB connection and uses this to identify the appropriate response to USB requests. This type lists the various possible states for the USB driver state machine. Some of these states are software defined and have no connection with any USB bus conditions.

Enumerator

CY_U3P_USB_INACTIVE USB module not started.

CY_U3P_USB_STARTED USB module started and waiting for configuration.

CY_U3P_USB_WAITING_FOR_DESC USB module waiting for a configuration descriptor.

CY_U3P_USB_CONFIGURED USB module completely configured.

CY_U3P_USB_VBUS_WAIT USB module waiting for Vbus to connect to USB host.

CY_U3P_USB_CONNECTED Waiting for USB connection.

CY_U3P_USB_ESTABLISHED USB connection active.

CY_U3P_USB_MS_ACTIVE Mass storage function active.

CY_U3P_USB_MAX_STATE Sentinel state.

5.32.6.6 enum CyU3PUSBSetDescType_t

Virtual descriptor types to be used to set descriptors.

Description

The user application can register different device and configuration descriptors with the USB driver, to be used at various USB connection speeds. To support this, the CyU3PUsbSetDesc call accepts a descriptor type parameter that is different from the USB standard descriptor type value that is embedded in the descriptor itself. This enumerated type is to be used by the application to register various descriptors with the USB driver.

See Also

CyU3PUsbSetDesc

Enumerator

CY_U3P_USB_SET_SS_DEVICE_DESCR SuperSpeed (USB 3.0) device descriptor.

CY_U3P_USB_SET_HS_DEVICE_DESCR USB 2.0 device descriptor.

CY_U3P_USB_SET_DEVQUAL_DESCR Device qualifier descriptor

CY_U3P_USB_SET_FS_CONFIG_DESCR Full Speed configuration descriptor.

CY_U3P_USB_SET_HS_CONFIG_DESCR High Speed configuration descriptor.

CY_U3P_USB_SET_STRING_DESCR String descriptor.

CY_U3P_USB_SET_SS_CONFIG_DESCR SuperSpeed configuration descriptor.

CY_U3P_USB_SET_SS_BOS_DESCR BOS descriptor.

CY_U3P_USB_SET_SS_DEVICE_DESCR Device descriptor for SuperSpeed.

CY_U3P_USB_SET_HS_DEVICE_DESCR Device descriptor for Hi-Speed and Full Speed.

CY_U3P_USB_SET_DEVQUAL_DESCR Device Qualifier descriptor.

CY_U3P_USB_SET_FS_CONFIG_DESCR Configuration descriptor for Full Speed.

CY_U3P_USB_SET_HS_CONFIG_DESCR Configuration descriptor for Hi-Speed.

CY_U3P_USB_SET_STRING_DESCR String Descriptor

CY_U3P_USB_SET_SS_CONFIG_DESCR Configuration descriptor for SuperSpeed.

CY_U3P_USB_SET_SS_BOS_DESCR BOS descriptor.

CY_U3P_USB_SET_OTG_DESCR OTG descriptor.

5.32.6.7 enum CyU3PUSBSpeed_t

List of supported USB connection speeds.

Enumerator

CY_U3P_NOT_CONNECTED USB device not connected.

CY_U3P_FULL_SPEED USB full speed.

CY_U3P_HIGH_SPEED High speed.

CY_U3P_SUPER_SPEED Super speed.

5.32.7 Function Documentation

5.32.7.1 CyU3PReturnStatus_t CyU3PConnectState (CyBool_t *connect*, CyBool_t *ssEnable*)

Enable / disable the USB connection.

Description

This function is used to enable or disable the USB PHYs on the FX3 device and to control the connection to the USB host in that manner. Re-enumeration can be achieved by disabling and then enabling the USB connection.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_ALREADY_STARTED - if the connection is already enabled.

CY_U3P_ERROR_NOT_CONFIGURED - if all of the necessary configuration has not been completed.

See Also

CyU3PUsbStart
CyU3PUsbStop
CyU3PUsbSetDesc
CyU3PGetConnectState

Parameters

connect	CyTrue to enable connection, CyFalse to disable connection.
ssEnable	Whether to enumerate as a SS device or FS/HS device

5.32.7.2 CyBool_t CyU3PGetConnectState (void)

Check whether the FX3 device is currently connected to a host.

Description

This function checks whether the the FX3 device is connected to a USB host. This depends on the detection of a valid Vbus input to the device.

Return value

CyTrue: The USB PHY on FX3 has been turned ON. CyFalse: The USB PHY on FX3 is turned OFF.

See Also

CyU3PUsbStart
CyU3PUsbStop
CyU3PConnectState

5.32.7.3 CyU3PReturnStatus_t CyU3PSetEpConfig (uint8_t *ep*, CyU3PEpConfig_t * *epinfo*)

Configure a USB endpoint.

Description

The FX3 device has 30 user configurable endpoints (1-OUT to 15-OUT and 1-IN to 15-IN) which can be separately selected and configured with desired properties such as endpoint type, the maximum packet size, amount of desired buffering. For bulk endpoints under super-speed, the number of bulk streams supported on that endpoint also needs to be specified.

All of these endpoints are kept disabled by default. This function is used to enable and set the properties for a specified endpoint. Separate calls need to be made to enable and configure each endpoint that needs to be used. The same function can be called to disable the endpoints after use.

Note

Please note that a mult (isoPkts) setting of greater than 0 (multiple packets or bursts in a micro-frame) is only supported for endpoints 3 and 7 in either direction.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_NULL_POINTER - when the epinfo pointer is NULL.

CY_U3P_ERROR_BAD_ARGUMENT - when the packet size or the endpoint number is invalid.

CY_U3P_ERROR_INVALID_CONFIGURATION - when a non-zero isoPkts value is used with endpoints other than 3 and 7.

See Also

CyU3PEpConfig_t

Parameters

ep	Endpoint number to be configured.
epinfo	Properties to associate with the endpoint.

5.32.7.4 CyU3PReturnStatus_t CyU3PSetEpPacketSize (uint8_t *ep*, uint16_t *maxPktSize*)

Set the maximum packet size for a USB endpoint.

Description

The DMA channels associated with USB endpoints on the FX3 device can be configured to have buffers of any user specified size (up to 64 KB). The FX3 device will allow multiple packets worth of data to be collected into a single DMA buffer as long as all of the packets are full packets. Any short packet arriving from USB will cause a DMA buffer to be wrapped up and committed to the consumer.

This API is used to set the maximum packet size that applies to the current endpoint. This needs to be set by the application based on the active USB connection speed, if there is a need to collect multiple data packets into a buffer. By default, this is set to the maximum packet size computed based on the user specified CyU3PEpConfig_t.pcktSize value, the USB connection speed and endpoint type.

Note

Please note that the maximum packet size will be automatically adjusted to the connection speed when the 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.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - if the endpoint or size specified is invalid.

See Also

CyU3PSetEpConfig
CyU3PUsbSetEpPktMode

Parameters

ep	Endpoint number to be configured.
maxPktSize	Maximum packet size for endpoint in bytes.

5.32.7.5 void CyU3PUsbAckSetup (void)

Complete the status handshake of a USB control request.

Description

This function is used to complete the status handshake of a USB control request that does not involve any data transfer. If there is a need for OUT or IN data transfers to process the control request, the CyU3PUsbGetEP0Data and CyU3PUsbSendEP0Data calls should be used instead.

This function should only be used if a positive ACK is to be sent to the USB host. To indicate an error condition, the CyU3PUsbStall call should be used to stall the endpoint EP0-OUT or EP0-IN.

Return value

None

See Also

CyU3PUSBSetupCb_t
CyU3PUsbRegisterSetupCallback
CyU3PUsbSendEP0Data
CyU3PUsbGetEP0Data

5.32.7.6 CyU3PReturnStatus_t CyU3PUsbChangeMapping (uint8_t *ep*, uint8_t *socketNum*, CyBool_t *remap*, uint16_t *newstreamId*, uint8_t *newep*)

Map a socket to stream of the specified endpoint.

Description

This function is used to map the stream of the specified endpoint to the socket passed as a parameter, this can be done if the socket is not already mapped.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if invalid parameter is passed to function.

See Also

CyU3PSetEpConfig
CyU3PUsbMapStream

Parameters

ep	Endpoint number to which the stream was mapped
socketNum	Socket number to which the stream was mapped
remap	CyTrue: if remapping is to be done
newstreamId	Stream id of the new stream to be mapped to the socket
newep	Endpoint number of the new stream to be mapped to the socket

5.32.7.7 CyU3PReturnStatus_t CyU3PUsbControlUsb2Support (CyBool_t *enable*)

Enable/Disable USB 2.0 device operation on FX3 device.

Description

By default, the FX3 device is configured to attempt both USB 3.0 and 2.0 connections to a USB host. The USB driver in the firmware automatically attempts a hi-speed or full speed connection, if the SuperSpeed connection fails to go through link training.

Some designs may use the FX3 device in USB 3.0 mode alone, and require that an external controller be used in USB 2.0 mode. This API can be used to modify the driver behavior to suppress the USB 2.0 device operation on FX3. USB events are provided to notify the user that the 3.0 connection has failed, and a 2.0 connection needs to be attempted. The CyU3PConnectState API should be called repeatedly to retry the USB 3.0 connection.

Return value

CY_U3P_SUCCESS if the configuration is updated properly.

CY_U3P_ERROR_NOT_STARTED if the USB block has not been started.

CY_U3P_ERROR_INVALID_SEQUENCE if the USB connection has already been enabled.

Parameters

enable	Whether USB 2.0 support on FX3 is enabled (default = enabled).

5.32.7.8 CyU3PReturnStatus_t CyU3PUsbControlVBusDetect (CyBool_t *enable,* CyBool_t *useVbatt*)

Enable/Disable VBus detection in FX3 firmware.

Description

When the FX3 device is functioning as a USB device, it performs Vbus detection by default; and will connect to the host only when VBus voltage is seen to be above 4.1 V. This API is used to configure the FX3 device to ignore the VBus input.

In some cases, the VBus voltage may be connected to the VBatt input of FX3 through a regulator. The firmware can be configured to automatically detect the VBatt input and enable/disable the USB connectivity accordingly. The useVbatt parameter can be set to select this mode of operation.

If the VBus signal from the USB connector is not connected to FX3 or connected to a GPIO, the user is expected to enable/disable the USB connection at appropriate times. In this case; both the enable and useVbatt parameters should be set to CyFalse, and the CyU3PConnectState API should be called directly by the user application.

Note

The USB block on the FX3 device draws power from the VBus supply by default. If the voltage source connected on the VBus pin is unable to provide enough power for this, the user should use the CyU3PUsbVBattEnable API to have the device draw power from VBatt instead.

Return value

CY_U3P_SUCCESS if the configuration is updated properly.

CY_U3P_ERROR_NOT_STARTED if the USB block is not started.

CY_U3P_ERROR_INVALID_SEQUENCE if the USB connection is already enabled.

See Also

CyU3PConnectState
CyU3PUsbVBattEnable

Parameters

enable	Whether VBus detection in firmware is enabled or disabled.
useVbatt	Whether VBatt should be used as an indication of VBus validity. This parameter is only relevant when the enable parameter is set to false.

5.32.7.9 CyU3PReturnStatus_t CyU3PUsbDoRemoteWakeup (void)

Trigger USB 2.0 Remote Wakeup signalling to the host.

Description

Self powered USB 2.0 devices which have been suspended can notify the host that they would like to be active again, by using remote wakeup signalling. This API call can be used to trigger remote wakeup signalling on the USB 2.0 pins of the FX3 device.

Return value

CY_U3P_SUCCESS - if the remote wakeup signalling was successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is not 2.0, is not suspended or remote wakeup is disabled.

5.32.7.10 void CyU3PUsbEnableEPPrefetch (void)

Configure the USB DMA interface to continuously pre-fetch data.

Description

Applications that use multiple USB IN endpoints and have high data rate may run into data corruption errors due to an underrun of data on the RAM to USB endpoint path. This API is used to configure the USB DMA interface to pre-fetch the data from RAM more aggressively, so as to prevent this kind of data corruption.

Receiving a CY_U3P_USB_EVENT_EP_UNDERRUN event from the USB driver is an indication that this API should be called by the application.

Return value

None

See Also

CY_U3P_USB_EVENT_EP_UNDERRUN

5.32.7.11 CyU3PReturnStatus_t CyU3PUsbEnableITPEvent (CyBool_t *enable*)

Function to enable/disable notification of SOF/ITP events to the application.

Description

Start Of Frame (SOF) and Isochronous Timestamp Packet (ITP) packets are sent by a USB host to connected devices on a periodic basis (every 125 us for high speed and super speed USB connections). While some applications may require notifications when these packets are received, they may cause unnecessary overhead in many other cases. This function is used to enable/disable notifications of SOF/ITP events from the FX3 firmware library to the application. These events are kept disabled by default and are sent through the regular USB event callback function.

Note

The SOF/ITP event notification is sent to the application in interrupt context so as to reduce latencies. Performing time consuming operations in these callbacks is therefore not recommended.

Return value

CY_U3P_SUCCESS - if the event change was successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

See Also

CyU3PUsbEventType_t
CyU3PUSBEventCb_t

Parameters

enable	Whether SOF/ITP event notification should be enabled.

5.32.7.12 void CyU3PUsbEpPrepare (CyU3PUSBSpeed_t *curSpeed*)

Prepare all enabled USB device endpoints for data transfer.

Description

This function prepares all of the enabled USB endpoints on the FX3 device for data transfer at the current link operating speed. This sets the maximum packet size for the endpoint based on the curSpeed parameter, and clears the sequence numbers, data toggled and any STALL conditions associated with these endpoints. This function also sets up the endpoint memory block on the FX3 device to remove the possibility of data underruns.

This API is called internally by the USB driver in response to USB connect, reset and SET_CONFIGURATION events; and does not generally need to be called directly by the user application.

Note

The caller of this function is expected to determine the current link operating speed, and then call this API with the appropriate parameter.

Return value

None

See Also

CyU3PUsbGetSpeed
CyU3PSetEpPacketSize
CyU3PUsbStall

Parameters

curSpeed	Current USB connection speed.

5.32.7.13 CyU3PReturnStatus_t CyU3PUsbEPSetBurstMode (uint8_t *ep*, CyBool_t *burstEnable*)

Function to enable burst mode operation for a USB endpoint.

Description

Under normal operation, the FX3 device will only send data from one DMA buffer as part of one burst transaction on an USB IN endpoint. In some cases, this constraint may limit the data transfer performance achieved. This limitation can be fixed by configuring the endpoints for burst mode, where the device will be combine data from multiple DMA buffers into a single burst transaction. This API is used to enable/disable burst mode on an USB endpoint.

Note

Setting up burst mode is not recommended for endpoints which are likely to transfer a significant number of ZLPs or short packets. This should ideally be used for cases where most of the packets transferred will be full packets.

Return value

CY_U3P_SUCCESS if the configuration was updated as required.

CY_U3P_ERROR_BAD_ARGUMENT if the endpoint specified is invalid.

Parameters

ep	The endpoint to be configured.
burstEnable	Whether burst mode is to be enabled or not.

5.32.7.14 CyU3PReturnStatus_t CyU3PUsbFlushEp (uint8_t *ep*)

Clear (flush) all data buffers associated with a specified endpoint.

Description

This function is used to clear the contents of all data buffers associated with a specified endpoint. This functionality is typically used to get rid of stale data when handling a USB bus reset or recovering an error/stall condition on the endpoint.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if invalid parameter is passed to function.

See Also

CyU3PUsbStall
CyU3PUsbResetEp
CyU3PUsbResetEndpointMemories

Parameters

ep	Endpoint number to be cleared.

5.32.7.15 CyU3PReturnStatus_t CyU3PUsbForceFullSpeed (CyBool_t *enable*)

Function to force the USB 2.0 device connection to full speed.

Description

When the FX3 is functioning as a USB device, the speed selection between super speed and other (2.0) speeds can be done through the CyU3PConnectState API call. On a 2.0 connection, the device uses the highest speed that the host supports; and this will be high speed in most cases. This API can be used to force the device to connect as a full-speed device instead of a high-speed device, and also to re-enable high speed operation.

Note

This API only prevents FX3 from connecting as a high speed device. If the ssEnable parameter to the CyU3PConnectState() API is set to True, the device may still connect as a super-speed device.

Return value

CY_U3P_SUCCESS - if the control operation was completed successfully.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection has already been enabled.

Parameters

enable	Whether to enable/disable the forced full speed operation.

5.32.7.16 CyU3PReturnStatus_t CyU3PUsbGetBooterVersion (uint8_t * *major_p*, uint8_t * *minor_p*, uint8_t * *patch_p*)

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

Description

The full FX3 firmware image could be loaded directly by the FX3 ROM boot-loader, or by a firmware application using the boot firmware library. This API can be used to fetch the version number of the boot firmware library that loaded this firmware image.

Return value

CY_U3P_SUCCESS if the boot firmware version could be identified and returned.

CY_U3P_ERROR_BAD_ARGUMENT if valid pointers are not provided for the Returns.

CY_U3P_ERROR_FAILURE if the boot firmware version could not be identified. This can happen if the firmware application was loaded directly by the boot loader, or if the noReenum option was not used.

Parameters

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

5.32.7.17 CyU3PReturnStatus_t CyU3PUsbGetDevProperty (CyU3PUsbDevProperty *type*, uint32_t * *buf*)

Function that returns some properties of the USB device.

Description

This function is used to retrieve properties of the USB device such as Device address, current ITP timestamp or frame number etc.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_NOT_CONFIGURED - when the USB device connection is not active.

CY_U3P_ERROR_BAD_ARGUMENT - when the property type being queried is invalid.

See Also

CyU3PUsbDevProperty

Parameters

type	The type of property to be queried.
buf	Buffer into which the property value is copied. The format of the data in the buffer depends on the property being queried. See CyU3PUsbDevProperty for details on how the property data is returned.

5.32.7.18 CyU3PReturnStatus_t CyU3PUsbGetEP0Data (uint16_t *count*, uint8_t * *buffer*, uint16_t * *readCount*)

Read data associated with a control-OUT transfer.

Description

This function is used to get the OUT data associated with a USB control transfer. The caller is responsible for ensuring that all of the data being sent by the host is retrieved. If the caller is not able to do this, the control request should be stalled using the CyU3PUsbStall API.

Multiple calls of this function can be made to fetch all of the data associated with a single control transfer, as long as each of the partial calls fetches an integral number of full data packets from the host.

If the control request is to be failed with a STALL handshake, the stall call has to be made before all of the OUT data has been read. The request will be completed with a positive ACK as soon as all of the OUT data has been received by the device.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer passed is NULL or a DMA failure occurs.

CY_U3P_XFER_CANCELLED - is returned if the transaction has already been cancelled.

CY_U3P_ERROR_TIMEOUT - if the scheduled data transfer is not completed within 5 seconds.

CY_U3P_ERROR_ABORTED - if the data transfer is aborted due to a USB reset or another control transfer.

CY_U3P_ERROR_DMA_FAILURE - if the data transfer fails due to a DMA error.

See Also

CyU3PUsbSendEP0Data
CyU3PUsbAckSetup
CyU3PUsbStall

Parameters

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).
buffer	Pointer to buffer where the data should be placed.
readCount	Output parameter which will be filled with the actual size of data read.

5.32.7.19 CyU3PReturnStatus_t CyU3PUsbGetEpCfg (uint8_t *ep*, CyBool_t * *isNak*, CyBool_t * *isStall*)

Retrieve the current state of the specified endpoint.

Description

This function retrieves the current NAK and STALL status of the specified endpoint. The isNak return value will be CyTrue if the endpoint is forced to NAK all requests. The isStall return value will be CyTrue if the endpoint is currently stalled.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if the EP specified is not valid.

See Also

CyU3PUsbSetEpNak
CyU3PUsbStall

Parameters

ep	Endpoint number to query.
isNak	Return parameter which will be filled with the NAK status.
isStall	Return parameter which will be filled with the STALL status.

5.32.7.20 CyU3PReturnStatus_t CyU3PUsbGetEpSeqNum (uint8_t *ep*, uint8_t * *seqnum_p*)

Get the current sequence number for an endpoint.

Description

This function is used to query the current sequence number for a USB endpoint. This API is only valid while the device is functioning at USB 3.0.

Return value

CY_U3P_SUCCESS if the query was successful and the sequence number is returned.

CY_U3P_ERROR_BAD_ARGUMENT if the ep or the seqnum_p parameter is invalid.

CY_U3P_ERROR_INVALID_SEQUENCE if the USB link is not at SuperSpeed.

See Also

CyU3PUsbSetEpSeqNum

Parameters

ep	Endpoint to be queried.
seqnum_p	Return parameter to be filled with sequence number.

5.32.7.21 CyU3PReturnStatus_t CyU3PUsbGetErrorCounts (uint16_t * *phy_err_cnt*, uint16_t * *lnk_err_cnt*)

Function to get the number of USB 3.0 PHY and LINK error counts detected by FX3.

Description

The FX3 device keeps track of the number of USB 3.0 PHY and LINK errors encountered during device operation. This API can be used to query the number of PHY and LINK errors detected since the last query was completed. Both error counts are cleared to zero whenever a query is performed.

The PHY errors tracked by FX3 are:

8b/10b Decode errors.

Elastic buffer overflow/underflow errors.

CRC errors.

Training sequence errors.

PHY lock loss.

The LINK errors tracked by FX3 are:

Header packet ACK timeout.

Link credit timeout.

Missing LGOOD_x / LCRD_x error.

Tx/Rx header sequence number errors.

Link power management timeout (missing LAU/LXU).

Link header sequence number / credit advertisement timeout.

Link header or LGO_x received before sequence number / credit advertisement.

Please note that both error counters saturate at a value of 0xFFFF.

Return value

CY_U3P_SUCCESS if the query API is successful.

CY_U3P_ERROR_BAD_ARGUMENT if valid pointers are not provided.

CY_U3P_ERROR_INVALID_SEQUENCE if the 3.0 connection is not active

Parameters

phy_err_cnt	Return parameter to be filled with phy error count.
lnk_err_cnt	Return parameter to be filled with link error count.

5.32.7.22 uint16_t CyU3PUsbGetEventLogIndex (void)

Get the current event log buffer index.

Description

This function is used to get the current write location within the USB event log buffer. This can be used to identify the most recent values that have been added to the buffer.

Return value

Current buffer index

See Also

CyU3PUsbInitEventLog

5.32.7.23 CyU3PReturnStatus_t CyU3PUsbGetLinkPowerState (CyU3PUsbLinkPowerMode * *mode_p*)

Function to get the current power state of the USB 3.0 link.

Description

This function is used to get the current power state of the USB 3.0 link. The actual link state will be returned through the mode_p parameter if the link is in any of the Ux states. If the link is in any of the other LTSSM states, CyU3PUsbLPM_Unknown will be returned. An error will be returned if the USB 3.0 connection is not enabled.

Return value

CY_U3P_SUCCESS - the link state has been successfully retrieved.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_NOT_CONFIGURED - if the USB connection is not active.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is not in 3.0 mode.

CY_U3P_ERROR_BAD_ARGUMENT - if a null pointer is passed in as the mode_p parameter.

See Also

CyU3PUsbLinkPowerMode
CyU3PUsbSetLinkPowerState

Parameters

mode_p	Return parameter that will be filled in with the current power state.

5.32.7.24 CyU3PUSBSpeed_t CyU3PUsbGetSpeed (void)

Get the connection speed at which USB is operating.

Description

This function is used to get the operating speed of the USB connection.

Return value

Current USB connection speed.

See Also

CyU3PUSBSpeed_t CyU3PConnectState CyU3PGetConnectState

5.32.7.25 void CyU3PUsbInitEventLog (uint8_t * *buffer*, uint32_t *bufSize*)

Function to initiate logging of USB state changes into a circular buffer.

Description

While debugging USB connection/transfer failures with FX3, it is useful to have a detailed log of the USB related state changes and events. This function is used to register a memory buffer into which the state change information will be logged by the USB driver. The buffer would be treated as a circular buffer into which the data is continuously logged.

Return value

None

See Also

CyU3PUsbAddToEventLog

Parameters

buffer	Pointer to memory buffer into which events are to be logged.
bufSize	Size of memory buffer in bytes.

5.32.7.26 CyBool_t CyU3PUsbIsStarted (void)

This function returns whether the USB device module has been started.

Description

Since there can be various modes of USB operations this API returns whether CyU3PUsbStart was invoked.

Return value

CyTrue if the USB module has been started.

CyFalse if the USB module is not running.

See Also

CyU3PUsbStart
CyU3PUsbStop

5.32.7.27 CyU3PReturnStatus_t CyU3PUsbJumpBackToBooter (uint32_t *address*)

Function to transfer control back to the FX3 2-stage bootloader.

Description

This function is used to transfer control back to the FX3 2-stage bootloader. The caller is expected to do cleanup of modules other than USB prior to this call. D-Cache needs to be cleaned before calling this function. The entry address for the booter firmware can be obtained by using the verbose option of the elf2img converter tool.

It is expected that all Serial Peripheral blocks are turned OFF before this function is called. Only the GPIO block can be left ON, if the booter had previously been configured to leave the GPIO block ON.

Return value

CY_U3P_ERROR_INVALID_SEQUENCE - If this function is called before the serial peripherals are turned off.

CY_U3P_ERROR_OPERN_DISABLED - If the FX3 2-stage booter doesn't support switching back.

Otherwise this is a non-returning call.

See Also

CyU3PUsbSetBooterSwitch

Parameters

address	Address of the FX3 2-stage bootloader's entry point.

5.32.7.28 CyU3PReturnStatus_t CyU3PUsbLPMDisable (void)

Function to prevent FX3 from entering U1/U2 states.

Description

The USB driver in FX3 firmware runs a state machine that determines whether entry to U1/U2 low power states is to be allowed. At most times, this decision is made automatically by the FX3 device itself based on whether it has any pending packets to go out. At other times, the decision is made by the firmware based on whether a 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.

This call also disables the acceptance of LPM-L1 entry requests when FX3 is functioning as a Hi-Speed or a full speed device.

Note

Indiscriminate use of this function can result in USB compliance test failures. Please ensure that the LPM functionality is enabled every time an USB connect or reset event is received. This can be disabled again after ensuring that the device is functioning in a performance critical mode.

Return value

CY_U3P_SUCCESS - if the operation is successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection has not been enabled.

See Also

CyU3PUsbLPMEnable

5.32.7.29 CyU3PReturnStatus_t CyU3PUsbLPMEnable (void)

Function to re-enable automated handling of U1/U2 state entry.

Description

This function is used to re-enable the USB driver state machine that governs the handling of U1/U2 requests from the USB host. This function removes the override that is specified using the CyU3PUsbLPMDisable call.

This function also enables the acceptance of LPM-L1 entry requests when FX3 is functioning as a Hi-Speed or a full speed device.

Note

It is expected that this call is made every time the application receives a USB connect or reset event, if a CyU3P-UsbLPMDisable call has been made previously.

Return value

CY_U3P_SUCCESS - if the operation is successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection has not been enabled.

See Also

CyU3PUsbLPMDisable

5.32.7.30 CyU3PReturnStatus_t CyU3PUsbMapStream (uint8_t *ep*, uint8_t *socketNum*, uint16_t *streamId*)

Map a socket to stream of the specified endpoint.

Description

This function is used to map the stream of the specified endpoint to the socket passed as a parameter, this can be done if the socket is not already mapped.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if invalid parameter is passed to function.

See Also

CyU3PSetEpConfig
CyU3PUsbChangeMapping

Parameters

ep	Endpoint number for which the stream is to be mapped.
socketNum	Socket number for mapping the stream.
streamId	StreamId to be mapped.

5.32.7.31 void CyU3PUsbRegisterEpEvtCallback (CyU3PUsbEpEvtCb_t *cbFunc*, uint32_t *eventMask*, uint16_t *outEpMask*, uint16_t *inEpMask*)

Register a callback function for notification of USB endpoint events.

Description

This function is used to register a callback function that will be invoked for notification of USB endpoint events during

device operation.

Return value

None

See Also

CyU3PUsbEpEvtCb_t
CyU3PUsbEpEvtType

Parameters

cbFunc	Callback function pointer.
eventMask	Bitmap variable representing the events that should be enabled.
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.
inEpMask	Bitmap variable representing the IN endpoints whose events are to be enabled.

5.32.7.32 void CyU3PUsbRegisterEventCallback (CyU3PUSBEventCb_t *callback*)

Register a USB event callback function.

Description

This function registers a USB event callback function with the USB driver. This function will be invoked by the driver every time an USB event of interest happens.

Return value

None

See Also

CyU3PUSBEventCb_t
CyU3PUsbEventType_t
CyU3PUsbStart

Parameters

callback	Event callback function pointer.

5.32.7.33 void CyU3PUsbRegisterLPMRequestCallback (CyU3PUsbLPMReqCb_t *cb*)

Register a USB 3.0 LPM request handler callback.

Description

This function is used to register a callback that handles Link Power state requests from the USB host. In the absence of a registered callback, all U1/U2 LPM entry requests will be failed by the FX3 USB driver.

Return value

None

See Also

CyU3PUsbLPMReqCb_t
CyU3PUsbSetLinkPowerState
CyU3PUsbGetLinkPowerState
CyU3PUsbLPMDisable
CyU3PUsbLPMEnable

Parameters

cb	Callback function pointer.

5.32.7.34 void CyU3PUsbRegisterSetupCallback (CyU3PUSBSetupCb_t *callback*, CyBool_t *fastEnum*)

Register a USB setup request handler.

Description

This function is used to register a USB setup request handler with the USB driver. The fastEnum parameter specifies whether this setup handler should be used only for unknown setup requests or for all USB setup requests.

Return value

None

See Also

CyU3PUSBSetupCb_t
CyU3PUsbStart

Parameters

callback	Setup request handler function.
fastEnum	Select fast enumeration mode by setting CyTrue.

5.32.7.35 CyU3PReturnStatus_t CyU3PUsbResetEndpointMemories (void)

Reset and re-initialize the endpoint memory block on FX3.

Description

Some transfer error conditions can leave the endpoint memories on FX3 in a bad state. This API is used to reset 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 value

CY_U3P_SUCCESS in case of successful reset and re-initialization.

CY_U3P_ERROR_NOT_STARTED if the USB block has not been started.

5.32.7.36 CyU3PReturnStatus_t CyU3PUsbResetEp (uint8_t *ep*)

Resets the specified endpoint.

Description

This function is used to reset USB endpoints while functioning at Super speed. The reset clears error conditions on the endpoint logic and prepares the endpoing for data transfer.

This function does nothing if called while in a USB 2.0 connection.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_BAD_ARGUMENT - when the endpoint number is invalid.

See Also

CyU3PUsbStall
CyU3PUsbFlushEp
CyU3PUsbResetEndpointMemories

Parameters

ep	Endpoint number to be cleared.

5.32.7.37 CyU3PReturnStatus_t CyU3PUsbSendDevNotification (uint8_t *notificationType*, uint32_t *param0*, uint32_t *param1*)

Function to send a DEV_NOTIFICATION Transaction Packet to the host.

Description

This API allows the user to send DEV_NOTIFICATION Transaction packets to the USB 3.0 host. The Notification Type and Notification Type Specific fields are accepted as parameters. None of the parameters are validated by the API, so the caller needs to ensure validity of these values.

Note

This API has to be called by the user after ensuring that the link is in U0 state. If the link is in U1/U2 and a TP has to be sent, the CyU3PUsbSetLinkPowerState API should be called first to trigger a recovery to U0.

Return value

CY_U3P_SUCCESS - if the DEV_NOTIFICATION TP was successfully send to the host.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is not in Super Speed mode or if the link is not in U0 state.

Parameters

notificationType	Notification type - No validation is performed by the API.
param0	Notification Type Specific Data - DWORD1[31:8].
param1	Notification Type Specific Data - DWORD2[31:0].

5.32.7.38 CyU3PReturnStatus_t CyU3PUsbSendEP0Data (uint16_t *count*, uint8_t * *buffer*)

Send data to the USB host via EP0-IN.

Description

This function is used to respond to USB control requests with an associated IN data transfer phase. The data in the buffer will be sent to the host in multiple packets of appropriate size as per the USB connection speed. Multiple calls of this function can be made to respond to a single control request as long as each call sends an integral number of full packets to the host. Any send call that results in a short packet will terminate the control transfer.

If the data in the buffer ends in a full packet, a zero length packet (ZLP) should be sent to the host to terminate the control transfer.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if the buffer passed is NULL or a DMA failure occurs.

CY_U3P_XFER_CANCELLED - is returned if the transaction has already been cancelled.

CY_U3P_ERROR_TIMEOUT - if the scheduled data transfer is not completed within 5 seconds.

CY_U3P_ERROR_ABORTED - if the data transfer is aborted due to a USB reset or another control transfer.

CY_U3P_ERROR_DMA_FAILURE - if the data transfer fails due to a DMA error.

See Also

CyU3PUSBSetupCb_t
CyU3PUsbGetEP0Data
CyU3PUsbAckSetup
CyU3PUsbStall

Parameters

count	The amount of data in bytes.
buffer	Pointer to buffer containing the data.

5.32.7.39 CyU3PReturnStatus_t CyU3PUsbSendErdy (uint8_t *ep*, uint16_t *bulkStream*)

Function to send an ERDY TP to a USB 3.0 host.

Description

This function allows the firmware to generate an ERDY TP for a specified endpoint. Under normal operation, ERDY TPs are automatically generated by the FX3 device as and when required. This function is provided to support exceptional cases where the firmware application needs to generate a specific ERDY TP.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is currently not in USB 3.0 mode.

CY_U3P_ERROR_BAD_ARGUMENT - if the endpoint specified is invalid.

See Also

CyU3PUsbSendNrdy

Parameters

ep	Endpoint for which the ERDY TP is to be generated. Bit 7 of the endpoint indicates direction.
bulkStream	Stream number for which the ERDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

5.32.7.40 CyU3PReturnStatus_t CyU3PUsbSendNrdy (uint8_t *ep*, uint16_t *bulkStream*)

Function to send an NRDY TP to a USB 3.0 host.

Description

This function allows the firmware to generate an NRDY TP for a specified endpoint. Under normal operation, NRDY TPs are automatically generated by the FX3 device as and when required. This function is provided to support exceptional cases where the firmware application needs to generate a specific NRDY TP.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is currently not in USB 3.0 mode.

CY_U3P_ERROR_BAD_ARGUMENT - if the endpoint specified is invalid.

See Also

CyU3PUsbSendErdy

Parameters

ep	Endpoint for which the NRDY TP is to be generated. Bit 7 of the endpoint indicates direction.
bulkStream	Stream number for which the NRDY TP is to be generated. Is only valid for stream enabled bulk endpoints and is ignored otherwise.

5.32.7.41 void CyU3PUsbSetBooterSwitch (CyBool_t *enable*)

Function to enable/disable switching control back to the FX3 2-stage bootloader.

Description

This function is used to enabled/disable the switching of control back to the FX3 2-stage bootloader. The function is expected to be called prior to CyU3PUsbJumpBackToBooter().

Return value

None

See Also

CyU3PUsbJumpBackToBooter
CyU3PUsbStart

Parameters

enable	CyTrue - Enable switching to booter; CyFalse - Disable switching to booter.

5.32.7.42 CyU3PReturnStatus_t CyU3PUsbSetDesc (CyU3PUSBSetDescType_t *desc_type*, uint8_t *desc_index*, uint8_t * *desc*)

Register a USB descriptor with the driver.

Description

This function is used to register a USB descriptor with the USB driver. The driver is capable of remembering one descriptor each of the various supported types as well as 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.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - when the buffer pointer is invalid.

CY_U3P_ERROR_BAD_DESCRIPTOR_TYPE - When the descriptor type is not valid.

CY_U3P_ERROR_BAD_INDEX - if the string descriptor index exceeds 16.

See Also

CyU3PUSBSetDescType_t
CyU3PUsbRegisterSetupCallback

Parameters

desc_type	Type of descriptor to register.
desc_index	Descriptor index: Used only for string descriptors.
desc	Pointer to buffer containing the descriptor.

5.32.7.43 CyU3PReturnStatus_t CyU3PUsbSetEpNak (uint8_t *ep*, CyBool_t *nak*)

Force or clear the NAK status on the specified endpoint.

Description

All bulk and interrupt endpoints on the FX3 device can be forced to NAK any host request while the corresponding function driver is not prepared for data transfers. This function is used to force the specified endpoint to NAK all requests, or to clear the NAK status and allow data transfers to proceed.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - if the EP specified is not valid.

See Also

CyU3PUsbGetEpCfg

Parameters

ep	Endpoint to be updated.
nak	CyTrue to force NAK, CyFalse to clear NAK.

5.32.7.44 CyU3PReturnStatus_t CyU3PUsbSetEpPktMode (uint8_t *ep*, CyBool_t *pktMode*)

Select whether a OUT endpoint will function in packet (one buffer per packet) mode or not.

Description

USB OUT endpoints on the FX3 device are setup by default to collect multiple full data packets into a single DMA buffer (size permitting). The DMA buffer into which data is received is wrapped up and made available to the consumer only if it is filled up, or if a short packet is received on the endpoint. The endpoint can also be configured for packet mode, where each DMA buffer will only hold one packet worth of data (regardless of the actual packet size) and gets wrapped up as soon as any data packet is received.

This API is used to switch the USB OUT endpoint between packet mode and transfer mode as required. This API only operates on non-control OUT endpoints.

See Also

CyU3PSetEpPacketSize

Return value

CY_U3P_SUCCESS if the endpoint mode was updated as requested.

CY_U3P_ERROR_NOT_STARTED if the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT if the endpoint specified is invalid.

Parameters

ep	Endpoint number to be configured.
pktMode	Whether the endpoint will function in packet mode or not.

5.32.7.45 CyU3PReturnStatus_t CyU3PUsbSetEpSeqNum (uint8_t *ep*, uint8_t *seqnum*)

Set the active sequence number for an endpoint.

Description

This function is used to set the active sequence number for a USB 3.0 endpoint to a desired value.

Return value

CY_U3P_SUCCESS if the update operation is successful.

CY_U3P_ERROR_BAD_ARGUMENT if the ep or seqnum parameter is invalid.

CY_U3P_ERROR_INVALID_SEQUENCE if the USB link is not at SuperSpeed.

See Also

CyU3PUsbGetEpSeqNum

Parameters

ep	Endpoint to be updated.
seqnum	Sequence number to be selected.

5.32.7.46 CyU3PReturnStatus_t CyU3PUsbSetLinkPowerState (CyU3PUsbLinkPowerMode *link_mode*)

Function to request a device entry into one of the U0, U1 or U2 link power modes.

Description

This function is used to request the FX3 USB driver to place the USB 3.0 link in a desired (U0, U1 or U2) power state. The U3 power state is not supported because U3 entry can only be triggered by the host. The request to move into U1 or U2 will only be allowed while the link is currently in the U0 state. The request to move into U0 will be allowed while the link is in the U1, U2 or U3 states.

Return value

CY_U3P_SUCCESS - when the requested link power state switch request has been initiated.

CY_U3P_ERROR_NOT_STARTED - if the USB driver has not been started.

CY_U3P_ERROR_NOT_CONFIGURED - if the USB connection is not active.

CY_U3P_ERROR_BAD_ARGUMENT - if the link state specified is invalid.

CY_U3P_ERROR_OPERN_DISABLED - if the USB connection is not in USB 3.0 mode or if the current link power state does not allow this transition.

See Also

CyU3PUsbLinkPowerMode
CyU3PUsbGetLinkPowerState
CyU3PUsbLPMEnable
CyU3PUsbLPMDisable

Parameters

link_mode	Desired link power state.

5.32.7.47 CyU3PReturnStatus_t CyU3PUsbSetTxDeemphasis (uint32_t *value*)

Set the Tx de-emphasis setting for the USB 3.0 signals.

Description

This API sets the Tx de-emphasis level used by FX3 on the USB 3.0 interface. Please use this API with caution. This API is expected to be called before calling the CyU3PConnectState() API to enable USB connections.

Return value

CY_U3P_SUCCESS if the setting is updated as expected.

CY_U3P_ERROR_BAD_ARGUMENT if the value specified is out of range.

Parameters

value	TX De-emphasis value. Should be less than 0x1F. Default value is 0x11.

5.32.7.48 CyU3PReturnStatus_t CyU3PUsbSetTxSwing (uint32_t *swing*)

Set the Tx amplitude range for the USB 3.0 signals.

Description

This API sets the Tx amplitude used by FX3 on the USB 3.0 interface. Please use this API with caution. The device has only been tested to work properly under the default swing setting of 0.9V (swing value set to 90). This API is expected to be called before calling the CyU3PConnectState() API to enable USB connections.

Return value

CY_U3P_SUCCESS if the setting is updated as expected.

CY_U3P_ERROR_BAD_ARGUMENT if the swing value specified is out of range.

Parameters

swing	TX Amplitude swing in 10 mV units. Should be less than 1.28V.

5.32.7.49 CyU3PReturnStatus_t CyU3PUsbStall (uint8_t *ep*, CyBool_t *stall*, CyBool_t *toggle*)

Set or clear the stall status of an endpoint.

Description

This function is to set or clear the stall status of a given endpoint. This function is to be used in response to SET_FEATURE and CLEAR_FEATURE requests from the host as well as for interface specific error handling. When the stall condition is being cleared, the data toggles for the endpoint can also be cleared. While an option is provided to leave the data toggles unmodified, this should only be used under specific conditions as recommended by Cypress.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_BAD_ARGUMENT - when the endpoint is invalid.

See Also

CyU3PSetEpConfig
CyU3PUsbGetEpCfg

Parameters

ep	Endpoint number to be modified.
stall	CyTrue: Set the stall condition, CyFalse: Clear the stall
toggle	CyTrue: Clear the data toggles in a Clear Stall call

5.32.7.50 CyU3PReturnStatus_t CyU3PUsbStart (void)

Start the USB driver.

Description

This function is used to start the USB device mode driver in the FX3 device and also to create the DMA channels required for the control endpoint.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_ALREADY_STARTED - when the USB driver has already been started.

CY_U3P_ERROR_CHANNEL_CREATE_FAILED - when the DMA channel creation for the control endpoint fails.

CY_U3P_ERROR_NO_REENUM_REQUIRED - if the USB block has been left running by the boot firmware image..

See Also

CyU3PUsbStop
CyU3PUsbIsStarted
CyU3PUsbVBattEnable
CyU3PUsbRegisterEventCallback
CyU3PUsbRegisterSetupCallback
CyU3PUsbRegisterEpEvtCallback
CyU3PUsbRegisterLPMRequestCallback
CyU3PUsbSetDesc
CyU3PConnectState

5.32.7.51 CyU3PReturnStatus_t CyU3PUsbStop (void)

Stop the USB driver.

Description

This function stops the USB driver on the FX3 device. It also frees up the DMA channels created for the control endpoint.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

Other DMA error codes - when the control endpoint DMA channel destroy fails.

See Also

CyU3PUsbStart
CyU3PUsbIsStarted

5.32.7.52 CyU3PReturnStatus_t CyU3PUsbVBattEnable (CyBool_t *enable*)

Configure USB block on FX3 to work off Vbatt power instead of Vbus.

Description

The USB block on the FX3 device can be configured to work off Vbus power or Vbatt power, with the Vbus power being the default setting. This function is used to enable/disable the Vbatt power input to the USB block.

This API needs to be called before the CyU3PConnectState API is called.

Return value

CY_U3P_SUCCESS - when the call is successful.

CY_U3P_ERROR_NOT_STARTED - when the USB driver has not been started.

CY_U3P_ERROR_INVALID_SEQUENCE - when the API is called after CyU3PConnectState has been called.

See Also

CyU3PUsbStart
CyU3PConnectState

Parameters

enable	CyTrue: Work off Vbatt, CyFalse: Work off Vbus.

5.33 firmware/u3p_firmware/inc/cyu3usbconst.h File Reference

This file defines constants that are derived from the USB specifications, for the use of the USB driver and user firmware.

```
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Macros

- #define **CY_U3P_USB_TARGET_MASK** (0x03) /* The Target mask */
- #define **CY_U3P_USB_TARGET_DEVICE** (0x00) /* The USB Target Device */
- #define **CY_U3P_USB_TARGET_INTF** (0x01) /* The USB Target Interface */
- #define **CY_U3P_USB_TARGET_ENDPT** (0x02) /* The USB Target Endpoint */
- #define **CY_U3P_USB_TARGET_OTHER** (0x03) /* The USB Target Other */
- #define **CY_U3P_USB_GS_DEVICE** (0x80) /* The standard device request, GET_STATUS device */
- #define **CY_U3P_USB_GS_INTERFACE** (0x81) /* The standard device request, GET_STATUS interface */
- #define **CY_U3P_USB_GS_ENDPOINT** (0x82) /* The standard device request, GET_STATUS endpoint */
- #define **CY_U3P_USB_TYPE_MASK** (0x60) /* The request type mask */
- #define **CY_U3P_USB_STANDARD_RQT** (0x00) /* The USB standard request */
- #define **CY_U3P_USB_CLASS_RQT** (0x20) /* The USB class request */
- #define **CY_U3P_USB_VENDOR_RQT** (0x40) /* The USB vendor request */
- #define **CY_U3P_USB_RESERVED_RQT** (0x60) /* The USB reserved request */
- #define **CY_U3P_USB3_TP_DEVADDR_POS** (25)
- #define **CY_U3P_USB3_TP_EPNUM_POS** (8)
- #define **CY_U3P_USB_OTG_STATUS_SELECTOR** (0xF000)

Typedefs

- typedef enum CyU3PUsbSetupCmds CyU3PUsbSetupCmds

  Standard device request codes.
- typedef enum CyU3PUsbEpType_t CyU3PUsbEpType_t

  Enumeration of the endpoint types.
- typedef enum CyU3PUsbDescType CyU3PUsbDescType

  Enumeration of descriptor types.
- typedef enum CyU3PUsbDevCapType CyU3PUsbDevCapType

  Device capability type codes.
- typedef enum CyU3PUsb3PacketType CyU3PUsb3PacketType

  USB 3.0 packet type codes.
- typedef enum CyU3PUsb3TpSubType CyU3PUsb3TpSubType

  USB 3.0 transaction packet sub type codes.
- typedef enum CyU3PUsbFeatureSelector CyU3PUsbFeatureSelector

  List of USB Feature selector codes.
- typedef enum CyU3PUsbLinkState_t CyU3PUsbLinkState_t

  Link state machine states.

Enumerations

- enum CyU3PUsbSetupCmds { CY_U3P_USB_SC_GET_STATUS = 0x00, CY_U3P_USB_SC_CLEAR_FEATURE, CY_U3P_USB_SC_RESERVED, CY_U3P_USB_SC_SET_FEATURE, CY_U3P_USB_SC_SET_ADDRESS = 0x05, CY_U3P_USB_SC_GET_DESCRIPTOR, CY_U3P_USB_SC_SET_DESCRIPTOR, CY_U3P_USB_SC_GET_CONFIGURATION, CY_U3P_USB_SC_SET_CONFIGURATION, CY_U3P_USB_SC_GET_INTERFACE, CY_U3P_USB_SC_SET_INTERFACE, CY_U3P_USB_SC_SYNC_FRAME, CY_U3P_USB_SC_SET_SEL = 0x30, CY_U3P_USB_SC_SET_ISOC_DELAY }

  Standard device request codes.
- enum CyU3PUsbEpType_t { CY_U3P_USB_EP_CONTROL = 0, CY_U3P_USB_EP_ISO = 1, CY_U3P_USB_EP_BULK = 2, CY_U3P_USB_EP_INTR = 3 }

  Enumeration of the endpoint types.
- enum CyU3PUsbDescType { CY_U3P_USB_DEVICE_DESCR = 0x01, CY_U3P_USB_CONFIG_DESCR, CY_U3P_USB_STRING_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_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, 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 }

  Enumeration of descriptor types.
- 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 }

  Device capability type codes.

- 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 }

  USB 3.0 packet type codes.

- enum CyU3PUsb3TpSubType { CY_U3P_USB3_TP_SUBTYPE_RES = 0, CY_U3P_USB3_TP_SUBTYPE_ACK, CY_U3P_USB3_TP_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 }

  USB 3.0 transaction packet sub type codes.

- enum CyU3PUsbFeatureSelector { CY_U3P_USBX_FS_EP_HALT = 0, CY_U3P_USB2_FS_REMOTE_WAKE = 1, CY_U3P_USB2_FS_TEST_MODE = 2, CY_U3P_USB2_OTG_B_HNP_ENABLE = 3, CY_U3P_USB2_OTG_A_HNP_SUPPORT = 4, CY_U3P_USB3_FS_U1_ENABLE = 48, CY_U3P_USB3_FS_U2_ENABLE = 49, CY_U3P_USB3_FS_LTM_ENABLE = 50 }

  List of USB Feature selector codes.

- enum CyU3PUsbLinkState_t { CY_U3P_UIB_LNK_STATE_SSDISABLED = 0x00, CY_U3P_UIB_LNK_STATE_RXDETECT_RES = 0x01, CY_U3P_UIB_LNK_STATE_RXDETECT_ACT = 0x02, CY_U3P_UIB_LNK_STATE_RXDETECT_QUT = 0x03, CY_U3P_UIB_LNK_STATE_SSINACT_QUT = 0x04, CY_U3P_UIB_LNK_STATE_SSINACT_DET = 0x05, CY_U3P_UIB_LNK_STATE_POLLING_LFPS = 0x08, CY_U3P_UIB_LNK_STATE_POLLING_RxEQ = 0x09, CY_U3P_UIB_LNK_STATE_POLLING_ACT = 0x0A, CY_U3P_UIB_LNK_STATE_POLLING_IDLE = 0x0C, CY_U3P_UIB_LNK_STATE_U0 = 0x10, CY_U3P_UIB_LNK_STATE_U1 = 0x11, CY_U3P_UIB_LNK_STATE_U2 = 0x12, CY_U3P_UIB_LNK_STATE_U3 = 0x13, CY_U3P_UIB_LNK_STATE_COMP = 0x17, CY_U3P_UIB_LNK_STATE_RECOV_ACT = 0x18, CY_U3P_UIB_LNK_STATE_RECOV_CNFG = 0x19, CY_U3P_UIB_LNK_STATE_RECOV_IDLE = 0x1A }

  Link state machine states.

5.33.1 Detailed Description

This file defines constants that are derived from the USB specifications, for the use of the USB driver and user firmware.

5.33.2 Typedef Documentation

5.33.2.1 typedef enum CyU3PUsb3PacketType CyU3PUsb3PacketType

USB 3.0 packet type codes.

Description

The following are the various USB 3.0 packet types.

5.33.2.2 typedef enum CyU3PUsb3TpSubType CyU3PUsb3TpSubType

USB 3.0 transaction packet sub type codes.

Description

The following are the various packet sub-types transmitted during SuperSpeed operation.

5.33.2.3 typedef enum CyU3PUsbDescType CyU3PUsbDescType

Enumeration of descriptor types.

Description

A USB device is identified by the descriptors provided. The following are among the standard descriptors defined by the USB specification.

5.33.2.4 typedef enum CyU3PUsbDevCapType CyU3PUsbDevCapType

Device capability type codes.

Description

The following are the various extended capabilities of the USB device.

5.33.2.5 typedef enum CyU3PUsbEpType_t CyU3PUsbEpType_t

Enumeration of the endpoint types.

Description

There are four types of endpoints. This defines the behaviour of the endpoint. The control endpoint is a compulsory for any device whereas the other endpoints are used as per device requirement.

5.33.2.6 typedef enum CyU3PUsbFeatureSelector CyU3PUsbFeatureSelector

List of USB Feature selector codes.

Description

The following are the various features that can be selected using the SetFeature request or cleared using Clear-Feature setup request.

5.33.2.7 typedef enum CyU3PUsbLinkState_t CyU3PUsbLinkState_t

Link state machine states.

Description

The following are the USB 3.0 link states of interest to FX3 firmware.

5.33.2.8 typedef enum CyU3PUsbSetupCmds CyU3PUsbSetupCmds

Standard device request codes.

Description

These are the various standard requests received from the USB host. The device is expected to respond to all these requests.

5.33.3 Enumeration Type Documentation

5.33.3.1 enum CyU3PUsb3PacketType

USB 3.0 packet type codes.

Description

The following are the various USB 3.0 packet types.

Enumerator

CY_U3P_USB3_PACK_TYPE_LMP Link Management Packet.

CY_U3P_USB3_PACK_TYPE_TP Transaction Packet.

CY_U3P_USB3_PACK_TYPE_DPH Data Packet Header.

CY_U3P_USB3_PACK_TYPE_ITP Isochronous Timestamp Packet.

5.33.3.2 enum CyU3PUsb3TpSubType

USB 3.0 transaction packet sub type codes.

Description

The following are the various packet sub-types transmitted during SuperSpeed operation.

Enumerator

CY_U3P_USB3_TP_SUBTYPE_RES Reserved.

CY_U3P_USB3_TP_SUBTYPE_ACK ACK TP.

CY_U3P_USB3_TP_SUBTYPE_NRDY NRDY TP.

CY_U3P_USB3_TP_SUBTYPE_ERDY ERDY TP.

CY_U3P_USB3_TP_SUBTYPE_STATUS STATUS TP.

CY_U3P_USB3_TP_SUBTYPE_STALL STALL TP.

CY_U3P_USB3_TP_SUBTYPE_NOTICE DEV_NOTIFICATION TP.

CY_U3P_USB3_TP_SUBTYPE_PING PING TP.

CY_U3P_USB3_TP_SUBTYPE_PINGRSP PING RESPONSE TP.

5.33.3.3 enum CyU3PUsbDescType

Enumeration of descriptor types.

Description

A USB device is identified by the descriptors provided. The following are among the standard descriptors defined by the USB specification.

Enumerator

CY_U3P_USB_DEVICE_DESCR Super Speed Device descr

CY_U3P_USB_CONFIG_DESCR Configuration

CY_U3P_USB_STRING_DESCR String

CY_U3P_USB_INTRFC_DESCR Interface

CY_U3P_USB_ENDPNT_DESCR End Point

CY_U3P_USB_DEVQUAL_DESCR Device Qualifier

CY_U3P_USB_OTHERSPEED_DESCR Other Speed Configuration

CY_U3P_USB_INTRFC_POWER_DESCR Interface power descriptor

CY_U3P_BOS_DESCR BOS descriptor

CY_U3P_DEVICE_CAPB_DESCR Device Capability descriptor

CY_U3P_USB_HID_DESCR HID descriptor

CY_U3P_USB_REPORT_DESCR Report descriptor

CY_U3P_SS_EP_COMPN_DESCR End Point companion descriptor

CY_U3P_USB_DEVICE_DESCR Device descriptor

CY_U3P_USB_CONFIG_DESCR Configuration descriptor

CY_U3P_USB_STRING_DESCR String descriptor

CY_U3P_USB_INTRFC_DESCR Interface descriptor

CY_U3P_USB_ENDPNT_DESCR Endpoint descriptor

CY_U3P_USB_DEVQUAL_DESCR Device Qualifier descriptor

CY_U3P_USB_OTHERSPEED_DESCR Other Speed Configuration descriptor

CY_U3P_USB_INTRFC_POWER_DESCR Interface power descriptor descriptor

CY_U3P_USB_OTG_DESCR OTG descriptor

CY_U3P_BOS_DESCR BOS descriptor

CY_U3P_DEVICE_CAPB_DESCR Device Capability descriptor

CY_U3P_USB_HID_DESCR HID descriptor

CY_U3P_USB_REPORT_DESCR Report descriptor

CY_U3P_SS_EP_COMPN_DESCR Endpoint companion descriptor

5.33.3.4 enum CyU3PUsbDevCapType

Device capability type codes.

Description

The following are the various extended capabilities of the USB device.

Enumerator

CY_U3P_WIRELESS_USB_CAPB_TYPE Wireless USB specific device level capabilities.

CY_U3P_USB2_EXTN_CAPB_TYPE USB 2.0 extension descriptor.

CY_U3P_SS_USB_CAPB_TYPE Super speed USB specific device level capabilities.

CY_U3P_CONTAINER_ID_CAPB_TYPE Unique ID used to identify the instance across all operating modes.

5.33.3.5 enum CyU3PUsbEpType_t

Enumeration of the endpoint types.

Description

There are four types of endpoints. This defines the behaviour of the endpoint. The control endpoint is a compulsory for any device whereas the other endpoints are used as per device requirement.

Enumerator

CY_U3P_USB_EP_CONTROL Control Endpoint Type

CY_U3P_USB_EP_ISO Isochronous Endpoint Type

CY_U3P_USB_EP_BULK Bulk Endpoint Type

CY_U3P_USB_EP_INTR Interrupt Endpoint Type

5.33.3.6 enum CyU3PUsbFeatureSelector

List of USB Feature selector codes.

Description

The following are the various features that can be selected using the SetFeature request or cleared using Clear-Feature setup request.

Enumerator

CY_U3P_USBX_FS_EP_HALT USB Endpoint HALT feature. Sets or clears EP stall.

CY_U3P_USB2_FS_REMOTE_WAKE USB 2.0 Remote Wakeup.

CY_U3P_USB2_FS_TEST_MODE USB 2.0 Test mode.

CY_U3P_USB2_OTG_B_HNP_ENABLE USB 2.0 OTG HNP enable signal to B-device.

CY_U3P_USB2_OTG_A_HNP_SUPPORT USB 2.0 OTG HNP supported indication to B-device.

CY_U3P_USB3_FS_U1_ENABLE USB 3.0 U1 Enable.

CY_U3P_USB3_FS_U2_ENABLE USB 3.0 U2 Enable.

CY_U3P_USB3_FS_LTM_ENABLE USB 3.0 LTM Enable.

5.33.3.7 enum CyU3PUsbLinkState_t

Link state machine states.

Description

The following are the USB 3.0 link states of interest to FX3 firmware.

Enumerator

CY_U3P_UIB_LNK_STATE_SSDISABLED SS.Disabled

CY_U3P_UIB_LNK_STATE_RXDETECT_RES Rx.Detect.Reset

CY_U3P_UIB_LNK_STATE_RXDETECT_ACT Rx.Detect.Active

CY_U3P_UIB_LNK_STATE_RXDETECT_QUT Rx.Detect.Quiet

CY_U3P_UIB_LNK_STATE_SSINACT_QUT SS.Inactive.Quiet

CY_U3P_UIB_LNK_STATE_SSINACT_DET SS.Inactive.Disconnect.Detect

CY_U3P_UIB_LNK_STATE_POLLING_LFPS Polling.LFPS

CY_U3P_UIB_LNK_STATE_POLLING_RxEQ Polling.RxEq

CY_U3P_UIB_LNK_STATE_POLLING_ACT Polling.Active

CY_U3P_UIB_LNK_STATE_POLLING_IDLE Polling.Idle

CY_U3P_UIB_LNK_STATE_U0 U0 - Active state

CY_U3P_UIB_LNK_STATE_U1 U1

CY_U3P_UIB_LNK_STATE_U2 U2

CY_U3P_UIB_LNK_STATE_U3 U3 - Suspend state

CY_U3P_UIB_LNK_STATE_COMP Compliance

CY_U3P_UIB_LNK_STATE_RECOV_ACT Recovery.Active

CY_U3P_UIB_LNK_STATE_RECOV_CNFG Recovery.Configuration

CY_U3P_UIB_LNK_STATE_RECOV_IDLE Recovery.Idle

5.33.3.8 enum CyU3PUsbSetupCmds

Standard device request codes.

Description

These are the various standard requests received from the USB host. The device is expected to respond to all these requests.

Enumerator

CY_U3P_USB_SC_GET_STATUS 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 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 Set system exit latency.

CY_U3P_USB_SC_SET_ISOC_DELAY Set isochronous delay.

5.34 firmware/u3p_firmware/inc/cyu3usbhost.h File Reference

The FX3 device supports programmable USB host implementation for a single USB host port at USB-HS, USB-FS and USB-LS speeds. The control pipe as well as the data pipes to be used can be configured through a set of USB host mode APIs. The USB host mode APIs also provide the capability to manage the host port.

```
#include <cyu3types.h>
#include <cyu3usbconst.h>
#include <cyu3externcstart.h>
#include <cyu3externcend.h>
```

Data Structures

- struct CyU3PUsbHostEpConfig_t

  Host mode endpoint configuration structure.
- struct CyU3PUsbHostConfig_t

  USB host mode configuration information.

Macros

- #define CY_U3P_USB_HOST_EPS_EPNUM_POS (0)

  Position of endpoint number field in the status word.
- #define CY_U3P_USB_HOST_EPS_EPNUM_MASK (0x0000000F)

  Mask for the endpoint number field in the status word.
- #define CY_U3P_USB_HOST_EPS_EPDIR (0x00000010)

  Endpoint direction bit (1 - OUT, 0 - IN).
- #define CY_U3P_USB_HOST_EPS_ACTIVE (0x00000020)

  Endpoint active bit. Indicates whether the EP is still active.
- #define CY_U3P_USB_HOST_EPS_HALT (0x00000040)

  Endpoint halt bit. Set if the endpoint is halted (stalled).
- #define CY_U3P_USB_HOST_EPS_OVER_UNDER_RUN (0x00000080)

  This bit is set if an overrun or underrun has happened on this endpoint.
- #define CY_U3P_USB_HOST_EPS_BABBLE (0x00000100)

  This bit is set if a babble was detected during the transfer.

- #define CY_U3P_USB_HOST_EPS_XACT_ERROR (0x00000200)

  This bit is set if there was a transfer error.
- #define CY_U3P_USB_HOST_EPS_PING (0x00000400)

  This bit is set if there was a PING packet.
- #define CY_U3P_USB_HOST_EPS_PHY_ERROR (0x00000800)

  This bit is set if there was a PHY error during the transfer.
- #define CY_U3P_USB_HOST_EPS_PID_ERROR (0x00001000)

  This bit is set if there was a PID error during the transfer.
- #define CY_U3P_USB_HOST_EPS_TIMEOUT_ERROR (0x00002000)

  This bit is set if there was a timeout error during the transfer.
- #define CY_U3P_USB_HOST_EPS_ISO_ORUN_ERROR (0x00004000)

  This bit is set if there was an ISO overrun error during the transfer.
- #define CY_U3P_USB_HOST_EPS_IOC_INT (0x00008000)

  This bit is set if there was an IOC interrupt.
- #define CY_U3P_USB_HOST_EPS_BYTE_COUNT_POS (16)

  Position of the remaining byte count field.
- #define CY_U3P_USB_HOST_EPS_BYTE_COUNT_MASK (0xFFFF0000)

  Mask for the remaining byte count field.
- #define CY_U3P_USB_HOST_PORT_STAT_CONNECTED (0x0001)

  This bit is set if a peripheral is connected downstream.
- #define CY_U3P_USB_HOST_PORT_STAT_ENABLED (0x0002)

  This bit is set if the host port has been enabled by the user.
- #define CY_U3P_USB_HOST_PORT_STAT_ACTIVE (0x0003)

  Mask to identify whether the port is active. A port is active if it is enabled and connected to a downstream peripheral.
- #define CY_U3P_USB_HOST_PORT_STAT_SUSPENDED (0x0004)

  This bit is set if the port has been suspended.

Typedefs

- typedef enum CyU3PUsbHostEpXferType_t CyU3PUsbHostEpXferType_t

  Mode of data transfer.
- typedef enum CyU3PUsbHostOpSpeed_t CyU3PUsbHostOpSpeed_t

  Speed of operation for the USB host port.
- typedef enum CyU3PUsbHostEventType_t CyU3PUsbHostEventType_t

  List of USB host mode events.
- typedef uint32_t CyU3PUsbHostEpStatus_t

  Host mode endpoint status.
- typedef uint16_t CyU3PUsbHostPortStatus_t

  Host mode port status information.
- typedef struct CyU3PUsbHostEpConfig_t CyU3PUsbHostEpConfig_t

  Host mode endpoint configuration structure.
- typedef void(∗ CyU3PUsbHostEventCb_t)(CyU3PUsbHostEventType_t evType, uint32_t evData)

  Host mode event callback function.
- typedef void(∗ CyU3PUsbHostXferCb_t)(uint8_t ep, CyU3PUsbHostEpStatus_t epStatus)

  Host mode endpoint transfer complete callback function.
- typedef struct CyU3PUsbHostConfig_t CyU3PUsbHostConfig_t

  USB host mode configuration information.

Enumerations

- enum CyU3PUsbHostEpXferType_t { CY_U3P_USB_HOST_EPXFER_NORMAL = 0, CY_U3P_USB_HOST_EPXFER_SETUP_OUT_DATA, CY_U3P_USB_HOST_EPXFER_SETUP_IN_DATA, CY_U3P_USB_HOST_EPXFER_SETUP_NO_DATA }

  Mode of data transfer.
- enum CyU3PUsbHostOpSpeed_t { CY_U3P_USB_HOST_LOW_SPEED = 0, CY_U3P_USB_HOST_FULL_SPEED, CY_U3P_USB_HOST_HIGH_SPEED }

  Speed of operation for the USB host port.
- enum CyU3PUsbHostEventType_t { CY_U3P_USB_HOST_EVENT_CONNECT = 0, CY_U3P_USB_HOST_EVENT_DISCONNECT }

  List of USB host mode events.

Functions

- CyBool_t CyU3PUsbHostIsStarted (void)

  Check whether the USB host stack has been started.
- CyU3PReturnStatus_t CyU3PUsbHostStart (CyU3PUsbHostConfig_t *hostCfg)

  This function initializes the USB 2.0 host stack.
- CyU3PReturnStatus_t CyU3PUsbHostStop (void)

  This function de-initializes the USB host stack.
- CyU3PReturnStatus_t CyU3PUsbHostPortEnable (void)

  This function enables the USB host port on the FX3 device.
- CyU3PReturnStatus_t CyU3PUsbHostPortDisable (void)

  Disable the USB host port.
- CyU3PReturnStatus_t CyU3PUsbHostGetPortStatus (CyU3PUsbHostPortStatus_t *portStatus, CyU3PUsbHostOpSpeed_t *portSpeed)

  This function retrieves the current port status.
- CyU3PReturnStatus_t CyU3PUsbHostPortReset (void)

  This function resets the USB host port.
- CyU3PReturnStatus_t CyU3PUsbHostPortSuspend (void)

  Suspend the USB host port.
- CyU3PReturnStatus_t CyU3PUsbHostPortResume (void)

  Resume the previously suspended USB host port.
- CyU3PReturnStatus_t CyU3PUsbHostGetFrameNumber (uint32_t *frameNumber)

  Get the current frame number to be used.
- CyU3PReturnStatus_t CyU3PUsbHostGetDeviceAddress (uint8_t *devAddr)

  Get the current downstream peripheral address.
- CyU3PReturnStatus_t CyU3PUsbHostSetDeviceAddress (uint8_t devAddr)

  Set (update) the downstream peripheral address.
- CyU3PReturnStatus_t CyU3PUsbHostEpAdd (uint8_t ep, CyU3PUsbHostEpConfig_t *cfg)

  Add an endpoint to the scheduler.
- CyU3PReturnStatus_t CyU3PUsbHostEpRemove (uint8_t ep)

  Removes an endpoint from the scheduler.
- CyU3PReturnStatus_t CyU3PUsbHostEpReset (uint8_t ep)

  Reset an endpoint.
- CyU3PReturnStatus_t CyU3PUsbHostSendSetupRqt (uint8_t *setupPkt, uint8_t *buf_p)

  Perform a USB control (EP0) transfer.
- CyU3PReturnStatus_t CyU3PUsbHostEpSetXfer (uint8_t ep, CyU3PUsbHostEpXferType_t type, uint32_t count)

  Start a non-EP0 data transfer.

- CyU3PReturnStatus_t CyU3PUsbHostEp0BeginXfer (void)

  Kick off a low level control endpoint transfer.
- CyU3PReturnStatus_t CyU3PUsbHostEpAbort (uint8_t ep)

  Abort pending transfers on the selected endpoint.
- CyU3PReturnStatus_t CyU3PUsbHostEpWaitForCompletion (uint8_t ep, CyU3PUsbHostEpStatus_t *epStatus, uint32_t waitOption)

  Wait for the current endpoint transfer to complete.

5.34.1 Detailed Description

The FX3 device supports programmable USB host implementation for a single USB host port at USB-HS, USB-FS and USB-LS speeds. The control pipe as well as the data pipes to be used can be configured through a set of USB host mode APIs. The USB host mode APIs also provide the capability to manage the host port.

5.34.2 Typedef Documentation

5.34.2.1 typedef struct CyU3PUsbHostConfig_t CyU3PUsbHostConfig_t

USB host mode configuration information.

Description

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

See Also

CyU3PUsbHostStart

5.34.2.2 typedef struct CyU3PUsbHostEpConfig_t CyU3PUsbHostEpConfig_t

Host mode endpoint configuration structure.

Description

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

See Also

CyU3PUsbHostEpAdd

5.34.2.3 typedef uint32_t CyU3PUsbHostEpStatus_t

Host mode endpoint status.

Description

At the end of each data transfer, the transfer scheduler on the FX3 device returns a 32-bit status value. This status value has a number of fields that report the status of the data transfer.

See Also

CyU3PUsbHostXferCb_t

5.34.2.4 typedef enum CyU3PUsbHostEpXferType_t CyU3PUsbHostEpXferType_t

Mode of data transfer.

Description

This type lists the various data transfer modes for USB endpoints on the FX3 host. The NORMAL mode is to be used for all endpoints other than the control endpoint. For the control (EP0) endpoint, the mode needs to be set on a per-transfer basis depending on the type of the control request.

See Also

CyU3PUsbHostEpSetXfer

5.34.2.5 typedef void(∗ CyU3PUsbHostEventCb_t)(CyU3PUsbHostEventType_t evType,uint32_t evData)

Host mode event callback function.

Description

The event callback function is used to notify the user application about peripheral connect and disconnect events.

See Also

CyU3PUsbHostStart

5.34.2.6 typedef enum CyU3PUsbHostEventType_t CyU3PUsbHostEventType_t

List of USB host mode events.

Description

This type lists the possible USB host mode related events that are reported to the application via the event callback function.

See Also

CyU3PUsbHostEventCb_t

5.34.2.7 typedef enum CyU3PUsbHostOpSpeed_t CyU3PUsbHostOpSpeed_t

Speed of operation for the USB host port.

Description

The USB host on the FX3 device supports Low, Full and Hi-Speed operation. This type defines named constants for each of these connection types.

See Also

CyU3PUsbHostGetPortStatus

5.34.2.8 typedef uint16_t CyU3PUsbHostPortStatus_t

Host mode port status information.

Description

The port status returned by the library is a 16-bit value with multiple fields.

See Also

CyU3PUsbHostGetPortStatus

5.34.2.9 typedef void(∗ CyU3PUsbHostXferCb_t)(uint8_t ep,CyU3PUsbHostEpStatus_t epStatus)

Host mode endpoint transfer complete callback function.

Description

The transfer callback function is registered when the USB host stack is started, and is used by the driver to notify the application about transfer completion.

See Also

CyU3PUsbHostStart
CyU3PUsbHostEpSetXfer

5.34.3 Enumeration Type Documentation

5.34.3.1 enum CyU3PUsbHostEpXferType_t

Mode of data transfer.

Description

This type lists the various data transfer modes for USB endpoints on the FX3 host. The NORMAL mode is to be used for all endpoints other than the control endpoint. For the control (EP0) endpoint, the mode needs to be set on a per-transfer basis depending on the type of the control request.

See Also

CyU3PUsbHostEpSetXfer

Enumerator

CY_U3P_USB_HOST_EPXFER_NORMAL Normal transfer. All non-EP0 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.

5.34.3.2 enum CyU3PUsbHostEventType_t

List of USB host mode events.

Description

This type lists the possible USB host mode related events that are reported to the application via the event callback function.

See Also

CyU3PUsbHostEventCb_t

Enumerator

CY_U3P_USB_HOST_EVENT_CONNECT USB Connect event.

CY_U3P_USB_HOST_EVENT_DISCONNECT USB Disconnect event.

5.34.3.3 enum CyU3PUsbHostOpSpeed_t

Speed of operation for the USB host port.

Description

The USB host on the FX3 device supports Low, Full and Hi-Speed operation. This type defines named constants for each of these connection types.

See Also

CyU3PUsbHostGetPortStatus

Enumerator

CY_U3P_USB_HOST_LOW_SPEED Host port is operating in low speed mode.

CY_U3P_USB_HOST_FULL_SPEED Host port is operating in full speed mode.

CY_U3P_USB_HOST_HIGH_SPEED Host port is operating in high speed mode.

5.34.4 Function Documentation

5.34.4.1 CyU3PReturnStatus_t CyU3PUsbHostEp0BeginXfer (void)

Kick off a low level control endpoint transfer.

Description

This function enables the endpoint and starts the transfer for EP0. The setup packet must be queued on EP0 egress socket before this function is called.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_INVALID_SEQUENCE - The endpoint is not configured or the transfer is not setup.

See Also

CyU3PUsbHostEpSetXfer
CyU3PUsbHostEpAbort

5.34.4.2 CyU3PReturnStatus_t CyU3PUsbHostEpAbort (uint8_t *ep*)

Abort pending transfers on the selected endpoint.

Description

This function aborts any ongoing data transfer on the selected endpoint and deactivates it. The DMA channels are not touched by the API, and they need to be separately reset.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_BAD_ARGUMENT - The endpoint number is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not added.

See Also

CyU3PUsbHostEpSetXfer
CyU3PUsbHostEp0BeginXfer

Parameters

ep	Endpoint to abort.

5.34.4.3 CyU3PReturnStatus_t CyU3PUsbHostEpAdd (uint8_t *ep*, CyU3PUsbHostEpConfig_t * *cfg*)

Add an endpoint to the scheduler.

Description

The USB host block on the FX3 maintains a schedule based on which data transfers on various endpoints are initiated. The firmware application should identify the active set of endpoints on the downstream peripheral, and add the corresponding endpoints to the execution schedule.

The schedule parameters that are passed to this function depend on the values reported by the peripheral in the endpoint descriptors.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NULL_POINTER - The input pointer was NULL.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_ALREADY_STARTED - The endpoint is already added.

CY_U3P_ERROR_BAD_ARGUMENT - One or more of the input parameter fields is invalid.

See Also

CyU3PUsbHostEpRemove

Parameters

ep	Endpoint to be added to the transfer schedule.
cfg	Endpoint configuration parameters.

5.34.4.4 CyU3PReturnStatus_t CyU3PUsbHostEpRemove (uint8_t *ep*)

Removes an endpoint from the scheduler.

Description

This function can be called to remove an endpoint from the transfer schedule, once it is no longer in use by the application.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_BAD_ARGUMENT - The endpoint number is invalid.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not yet added.

See Also

CyU3PUsbHostEpAdd

Parameters

ep	Endpoint to be removed from scheduler.

5.34.4.5 CyU3PReturnStatus_t CyU3PUsbHostEpReset (uint8_t *ep*)

Reset an endpoint.

Description

This function flushes the internal buffers and resets the data toggles for an endpoint. This should only be called when there is no active transfer on the endpoint.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_BAD_ARGUMENT - The endpoint number is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not added.

CY_U3P_ERROR_INVALID_SEQUENCE - The endpoint is active.

See Also

CyU3PUsbHostEpAdd

Parameters

ep	Endpoint to be reset.

5.34.4.6 CyU3PReturnStatus_t CyU3PUsbHostEpSetXfer (uint8_t *ep,* CyU3PUsbHostEpXferType_t *type,* uint32_t *count*)

Start a non-EP0 data transfer.

Description

This function will enable the endpoint and setup the data transfer for all EPs except for EP0. This can also be used for EP0 when the low level control flag has been set to true.

In the case of a low level EP0 transfer, the 8 byte setup packet needs to be queued on the EP0 egress DMA channel. If there is a data phase, that also needs to be queued on the EP0 ingress or egress DMA channel. Once the DMA channels have been setup, the EP0 transfer can be kicked off by calling the CyU3PUsbHostEp0BeginXfer API.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_BAD_ARGUMENT - One or more input parameters is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not added.

CY_U3P_ERROR_INVALID_SEQUENCE - The endpoint is already active.

See Also

CyU3PUsbHostEp0BeginXfer
CyU3PUsbHostEpAbort

Parameters

ep	Endpoint to configure.
type	Type of transfer. This is meaningful only for EP0.
count	Size of data to be transferred.

5.34.4.7 CyU3PReturnStatus_t CyU3PUsbHostEpWaitForCompletion (uint8_t *ep*, CyU3PUsbHostEpStatus_t * *epStatus*, uint32_t *waitOption*)

Wait for the current endpoint transfer to complete.

Description

The function waits for the transfer to complete or get aborted. If this does not happen within the timeout specified, it returns CY_U3P_ERROR_TIMEOUT.

A timeout error does not necessarily mean that it has failed, and it is possible that another call to this API returns successfully.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_BAD_ARGUMENT - The endpoint number is invalid.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not added.

CY_U3P_ERROR_INVALID_SEQUENCE - No active transfer.

CY_U3P_ERROR_TIMEOUT - The transfer is not complete at the end of the specified timeout duration.

CY_U3P_ERROR_STALLED - The transfer was stalled by the USB peripheral.

See Also

CyU3PUsbHostEpSetXfer
CyU3PUsbHostEp0BeginXfer

Parameters

ep	Endpoint to wait on.
epStatus	Output parameter to be filled in with transfer status.
waitOption	Timeout duration to wait for.

5.34.4.8 CyU3PReturnStatus_t CyU3PUsbHostGetDeviceAddress (uint8_t * *devAddr*)

Get the current downstream peripheral address.

Description

This function returns the device address that has been assigned by FX3 to the downstream peripheral. A return value of zero indicates that the peripheral has been disconnected.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NULL_POINTER - The pointer provided is NULL.

CY_U3P_ERROR_NOT_STARTED - The host port is not enabled.

See Also

CyU3PUsbHostSetDeviceAddress

Parameters

devAddr	Pointer to load the current device address.

5.34.4.9 CyU3PReturnStatus_t CyU3PUsbHostGetFrameNumber (uint32_t * *frameNumber*)

Get the current frame number to be used.

Description

This function gets the current USB frame number. The frame number query is not synchronized with the scheduler and only returns the number at the time of the API call.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NULL_POINTER - The input pointer was NULL.

CY_U3P_ERROR_NOT_STARTED - The host port is not enabled.

See Also

CyU3PUsbHostStart
CyU3PUsbHostPortEnable

Parameters

frameNumber	Pointer to load the frame number.

5.34.4.10 CyU3PReturnStatus_t CyU3PUsbHostGetPortStatus (CyU3PUsbHostPortStatus_t * *portStatus*, CyU3PUsbHostOpSpeed_t * *portSpeed*)

This function retrieves the current port status.

Description

This function retrieves the current state and speed of operation of the USB host port on the FX3 device.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The host stack is not running.

See Also

CyU3PUsbHostPortEnable
CyU3PUsbHostPortDisable

5.34.4.11 CyBool_t CyU3PUsbHostIsStarted (void)

Check whether the USB host stack has been started.

Description

This function is used to check whether the USB host stack has been started.

Return value

CyTrue - USB host module started.

CyFalse - USB host module not started.

See Also

CyU3PUsbHostStart
CyU3PUsbHostStop

5.34.4.12 CyU3PReturnStatus_t CyU3PUsbHostPortDisable (void)

Disable the USB host port.

Description

This function disables the USB host port on the FX3 device. The device will still be able to detect a newly connected peripheral after the port is disabled.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

See Also

CyU3PUsbHostPortEnable
CyU3PUsbHostGetPortStatus

5.34.4.13 CyU3PReturnStatus_t CyU3PUsbHostPortEnable (void)

This function enables the USB host port on the FX3 device.

Description

The USB host port on the FX3 is kept disabled when the host stack is started. The driver uses the event callback to notify that a downstream peripheral has been connected, and then the application can call this API to enable the host port.

This function enables the port and sets it to the correct speed of operation. When the down-stream peripheral is disconnected, the port automatically gets disabled.

The FX3 device has only a single usb host port which can support a single peripheral device. Hubs are not supported.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The host stack is not running.

CY_U3P_ERROR_ALREADY_STARTED - The port is already enabled.

CY_U3P_ERROR_FAILURE - No downstream peripheral attached.

See Also

CyU3PUsbHostPortDisable
CyU3PUsbHostGetPortStatus

5.34.4.14 CyU3PReturnStatus_t CyU3PUsbHostPortReset (void)

This function resets the USB host port.

Description

This function resets the USB host port so that the peripheral connected can be re-enumerated. Calling this function is equivalent to a port disable call followed by a port enable call.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The host stack is not running.

See Also

CyU3PUsbHostPortEnable
CyU3PUsbHostPortDisable
CyU3PUsbHostGetPortStatus

5.34.4.15 CyU3PReturnStatus_t CyU3PUsbHostPortResume (void)

Resume the previously suspended USB host port.

Description

This function resumes the USB host port, after it was previously suspended through the CyU3PUsbHostPort-Suspend API.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled or device got disconnected.

CY_U3P_ERROR_INVALID_SEQUENCE - The port is not suspended.

See Also

CyU3PUsbHostPortSuspend
CyU3PUsbHostGetPortStatus

5.34.4.16 CyU3PReturnStatus_t CyU3PUsbHostPortSuspend (void)

Suspend the USB host port.

Description

This function suspends the USB host port. Please note that the FX3 device does not support remote wakeup and the port resumption has to be done by the firmware application.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The port is not active.

CY_U3P_ERROR_INVALID_SEQUENCE - There is an active data transfer.

See Also

CyU3PUsbHostPortResume
CyU3PUsbHostGetPortStatus

5.34.4.17 CyU3PReturnStatus_t CyU3PUsbHostSendSetupRqt (uint8_t * *setupPkt*, uint8_t * *buf_p*)

Perform a USB control (EP0) transfer.

Description

This function performs all of the operations associated with a USB control (EP0) transfer. This is only valid when the ep0LowLevelControl setting is false.

The API initiates the transfer of setup packet, but does not wait for completion. The setup, data and status phases of the transfer will be handled by the driver; and the transfer complete callback shall be called when all of these are done.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_SUPPORTED - EP0 Low level control enabled.

CY_U3P_ERROR_NOT_STARTED - The port is not enabled.

CY_U3P_ERROR_INVALID_SEQUENCE - The endpoint is already active.

CY_U3P_ERROR_NOT_CONFIGURED - The endpoint is not added.

CY_U3P_ERROR_DMA_FAILURE - Error in setting internal DMA channels.

See Also

CyU3PUsbHostStart
CyU3PUsbHostEpAbort

Parameters

setupPkt	Pointer to buffer containing the 8 byte setup packet.
buf_p	Buffer for the data phase. The buffer should be big enough to handle the complete data phase, and should already contain the data in the case of an OUT data transfer..

5.34.4.18 CyU3PReturnStatus_t CyU3PUsbHostSetDeviceAddress (uint8_t *devAddr*)

Set (update) the downstream peripheral address.

Description

This function sets (updates) the device address that should be used by the FX3 to talk to the downstream peripheral. This address is initialized to zero whenever the port is enabled. The application should set the address after completing a SET_ADDRESS command successfully.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_BAD_ARGUMENT - The address parameter is invalid.

CY_U3P_ERROR_NOT_STARTED - The host port is not enabled.

See Also

CyU3PUsbHostGetDeviceAddress

Parameters

devAddr	Device address to be set.

5.34.4.19 CyU3PReturnStatus_t CyU3PUsbHostStart (CyU3PUsbHostConfig_t * *hostCfg*)

This function initializes the USB 2.0 host stack.

Description

This function enables the USB block and configures it to function as a host. The configuration parameters cannot be changed unless the stack is stopped and re-started.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_SUPPORTED - if the current FX3 device does not support the USB 2.0 host.

CY_U3P_ERROR_NULL_POINTER - If NULL pointer is passed in as parameter.

CY_U3P_ERROR_ALREADY_STARTED - The host stack is already running.

CY_U3P_ERROR_INVALID_SEQUENCE - FX3 is in the wrong OTG mode or USB device stack is running.

See Also

CyU3PUsbHostConfig_t
CyU3PUsbHostStop
CyU3PUsbHostIsStarted

Parameters

hostCfg	Pointer to the host configuration information.

5.34.4.20 CyU3PReturnStatus_t CyU3PUsbHostStop (void)

This function de-initializes the USB host stack.

Description

This function disables the USB host mode operation and de-initializes the USB host stack.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The host mode stack is not running.

See Also

CyU3PUsbHostStart
CyU3PUsbHostIsStarted

5.35 firmware/u3p_firmware/inc/cyu3usbotg.h File Reference

The FX3 device supports USB 2.0 On-The-Go (OTG) functionality which allows the device to identify whether it should function as a USB host or a peripheral. This file defines the data types and APIs provided by the USB driver on FX3 for OTG management.

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

Data Structures

- struct CyU3POtgConfig_t

  OTG configuration information.

Macros

- #define CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL (10000)

  Maximum value in ms that can be used for the SRP repeat interval.

Typedefs

- typedef enum CyU3POtgMode_t CyU3POtgMode_t

  OTG modes of operation.
- typedef enum CyU3POtgChargerDetectMode_t CyU3POtgChargerDetectMode_t

  Various charger detection methods supported.
- typedef enum CyU3POtgEvent_t CyU3POtgEvent_t

  List of OTG events.
- typedef enum CyU3POtgPeripheralType_t CyU3POtgPeripheralType_t

  List of OTG peripheral types.
- typedef void(∗ CyU3POtgEventCallback_t)(CyU3POtgEvent_t event, uint32_t input)

  OTG event callback function.
- typedef struct CyU3POtgConfig_t CyU3POtgConfig_t

  OTG configuration information.

Enumerations

- enum CyU3POtgMode_t { CY_U3P_OTG_MODE_DEVICE_ONLY = 0, CY_U3P_OTG_MODE_HOST_ONLY, CY_U3P_OTG_MODE_OTG, CY_U3P_OTG_MODE_CARKIT_PPORT, CY_U3P_OTG_MODE_CARKIT_UART, CY_U3P_OTG_NUM_MODES }

  OTG modes of operation.
- enum CyU3POtgChargerDetectMode_t { CY_U3P_OTG_CHARGER_DETECT_ACA_MODE = 0, CY_U3P_OTG_CHARGER_DETECT_MOT_EMU, CY_U3P_OTG_CHARGER_DETECT_NUM_MODES }

  Various charger detection methods supported.
- enum CyU3POtgEvent_t { CY_U3P_OTG_PERIPHERAL_CHANGE = 0, CY_U3P_OTG_SRP_DETECT, CY_U3P_OTG_VBUS_VALID_CHANGE }

  List of OTG events.
- enum CyU3POtgPeripheralType_t { CY_U3P_OTG_TYPE_DISABLED = 0, CY_U3P_OTG_TYPE_A_CABLE, CY_U3P_OTG_TYPE_B_CABLE, CY_U3P_OTG_TYPE_ACA_A_CHG, CY_U3P_OTG_TYPE_ACA_B_CHG, CY_U3P_OTG_TYPE_ACA_C_CHG, CY_U3P_OTG_TYPE_MOT_MPX200, CY_U3P_OTG_TYPE_MOT_CHG, CY_U3P_OTG_TYPE_MOT_MID, CY_U3P_OTG_TYPE_MOT_FAST }

  List of OTG peripheral types.

Functions

- CyBool_t CyU3POtgIsStarted (void)

  Check whether the OTG module has been started.
- CyU3POtgPeripheralType_t CyU3POtgGetPeripheralType (void)

  Identify the type of USB peripheral attached to FX3.
- CyU3POtgMode_t CyU3POtgGetMode (void)

  Retrieve the currently selected OTG operating mode.
- CyBool_t CyU3POtgIsDeviceMode (void)

  Check whether USB device (peripheral) mode operation is permitted.
- CyBool_t CyU3POtgIsHostMode (void)

  Check whether USB host mode operation is permitted.
- CyU3PReturnStatus_t CyU3POtgStart (CyU3POtgConfig_t *cfg)

  Initialize the OTG module.
- CyU3PReturnStatus_t CyU3POtgStop (void)

  Disable the OTG module.
- CyU3PReturnStatus_t CyU3POtgSrpStart (uint32_t repeatInterval)

  Initiate an SRP request.
- CyU3PReturnStatus_t CyU3POtgSrpAbort (void)

  Abort SRP request.
- CyU3PReturnStatus_t CyU3POtgRequestHnp (CyBool_t isEnable)

  Set the HNP request bit in the device status word.
- CyU3PReturnStatus_t CyU3POtgHnpEnable (CyBool_t isEnable)

  Initiate a HNP role change.
- CyBool_t CyU3POtgIsHnpEnabled (void)

  Check if a role reversal mode is active or not.
- CyBool_t CyU3POtgIsVBusValid (void)

  Check if a valid VBus is available.

5.35.1 Detailed Description

The FX3 device supports USB 2.0 On-The-Go (OTG) functionality which allows the device to identify whether it should function as a USB host or a peripheral. This file defines the data types and APIs provided by the USB driver on FX3 for OTG management. **Description**

When the FX3 USB port is in the device mode of operation, it can function as per USB 3.0 specification, and function at SuperSpeed, Hi-Speed or full speed depending upon the host capabilities.

When configured as a USB host, the port supports USB 2.0 operations at Hi-Speed, full speed or low speed according to the peripheral capabilities. In host mode of operation, only a single peripheral can be attached at a time and HUB devices are not supported.

The OTG port on FX3 is also capable of ACA charger detection based on the USB Battery Charging specification 1.1 or Motorola EMU specification depending on the selected configuration. The peripheral type detection is based on the ID pin state. The OTG port supports D+ pulsing based SRP and also supports Host Negotiation Protocol (HNP).

5.35.2 Typedef Documentation

5.35.2.1 typedef enum CyU3POtgChargerDetectMode_t CyU3POtgChargerDetectMode_t

Various charger detection methods supported.

Description

The FX3 device can detect chargers based on standard ACA requirements as well as Motorola EMU (enhanced mini USB) requirements. This enumeration lists the various charger detect modes available on FX3.

Charger detection is based on the state of the OTG ID pin and so is available only in CY_U3P_OTG_MODE_OTG mode of operation.

See Also

CyU3POtgMode_t
CyU3POtgConfig_t

5.35.2.2 typedef struct CyU3POtgConfig_t CyU3POtgConfig_t

OTG configuration information.

Description

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

See Also

CyU3POtgMode_t
CyU3POtgEventCallback_t
CyU3POtgStart

5.35.2.3 typedef enum CyU3POtgEvent_t CyU3POtgEvent_t

List of OTG events.

Description

The enumeration lists the various OTG events that the USB driver provides to the user application.

See Also

CyU3POtgEventCallback_t
CyU3POtgPeripheralType_t

5.35.2.4 typedef void(∗ CyU3POtgEventCallback_t)(CyU3POtgEvent_t event,uint32_t input)

OTG event callback function.

Description

This type defines the prototype of the event callback function that is called by the USB driver to notify the application about OTG events.

The input to the callback is dependant on the actual event. For the CY_U3P_OTG_PERIPHERAL_CHANGE event, this specifies the type of peripheral detected (CyU3POtgPeripheralType_t). For the VBus change event, it specifies the status of VBus.

See Also

CyU3POtgConfig_t
CyU3POtgEvent_t
CyU3POtgPeripheralType_t

5.35.2.5 typedef enum CyU3POtgMode_t CyU3POtgMode_t

OTG modes of operation.

Description

The FX3 device has a single USB port which can function in multiple modes. It can act as a USB 2.0 host or as a USB 3.0 device. It can also route the USB 2.0 lines (D+/D-) to the UART block for car-kit mode of operation.

This enumeration lists the various modes of operation allowed for the device. The default mode of operation is CY_U3P_OTG_MODE_DEVICE_ONLY.

See Also

CyU3POtgConfig_t

5.35.2.6 typedef enum CyU3POtgPeripheralType_t CyU3POtgPeripheralType_t

List of OTG peripheral types.

Description

This enumeration lists the various types of OTG peripherals that can be detected by the FX3.

See Also

CyU3POtgEventCallback_t
CyU3POtgEvent_t

5.35.3 Enumeration Type Documentation

5.35.3.1 enum CyU3POtgChargerDetectMode_t

Various charger detection methods supported.

Description

The FX3 device can detect chargers based on standard ACA requirements as well as Motorola EMU (enhanced mini USB) requirements. This enumeration lists the various charger detect modes available on FX3.

Charger detection is based on the state of the OTG ID pin and so is available only in CY_U3P_OTG_MODE_OTG mode of operation.

See Also

CyU3POtgMode_t
CyU3POtgConfig_t

Enumerator

CY_U3P_OTG_CHARGER_DETECT_ACA_MODE Charger detection is based on standard ACA charger mode. This is the default mode.

CY_U3P_OTG_CHARGER_DETECT_MOT_EMU Charger detection is based on Motorola Enhanced Mini USB (EMU) requirements.

CY_U3P_OTG_CHARGER_DETECT_NUM_MODES Number of charger detection modes.

5.35.3.2 enum CyU3POtgEvent_t

List of OTG events.

Description

The enumeration lists the various OTG events that the USB driver provides to the user application.

See Also

CyU3POtgEventCallback_t
CyU3POtgPeripheralType_t

Enumerator

CY_U3P_OTG_PERIPHERAL_CHANGE The OTG peripheral attached to FX3 has changed (removed or new device connected). The parameter to the event callback identifies the type of peripheral attached.

CY_U3P_OTG_SRP_DETECT The remote device has initiated an SRP request. On receiving this request, the user is expected to turn on the VBus.

CY_U3P_OTG_VBUS_VALID_CHANGE Notifies that VBus state has changed. The parameter is CyTrue if VBus is active and CyFalse if VBus is not active.

5.35.3.3 enum CyU3POtgMode_t

OTG modes of operation.

Description

The FX3 device has a single USB port which can function in multiple modes. It can act as a USB 2.0 host or as a USB 3.0 device. It can also route the USB 2.0 lines (D+/D-) to the UART block for car-kit mode of operation.

This enumeration lists the various modes of operation allowed for the device. The default mode of operation is CY_U3P_OTG_MODE_DEVICE_ONLY.

See Also

CyU3POtgConfig_t

Enumerator

CY_U3P_OTG_MODE_DEVICE_ONLY USB port acts in device only mode. The ID pin value is ignored, and charger detection is disabled.

CY_U3P_OTG_MODE_HOST_ONLY USB port acts in host only mode. The ID pin value is ignored, and charger detection is disabled.

CY_U3P_OTG_MODE_OTG USB port acts in OTG mode and identifies the mode of operation based on the state of the ID pin. This mode also enables charger detection based on the ID pin.

CY_U3P_OTG_MODE_CARKIT_PPORT The D+/D- lines are routed to the P-Port pads PIB_CTL11 and PIB_CTL12. Charger detection is disabled.

CY_U3P_OTG_MODE_CARKIT_UART The D+/D- lines are routed to the FX3 UART lines. FX3 UART will not function in this mode. Charger detection is disabled.

CY_U3P_OTG_NUM_MODES Number of OTG modes.

5.35.3.4 enum CyU3POtgPeripheralType_t

List of OTG peripheral types.

Description

This enumeration lists the various types of OTG peripherals that can be detected by the FX3.

See Also

CyU3POtgEventCallback_t
CyU3POtgEvent_t

Enumerator

CY_U3P_OTG_TYPE_DISABLED Device type not identified because the OTG mode detection is disabled.

CY_U3P_OTG_TYPE_A_CABLE OTG A-type peripheral cable connected to FX3. FX3 is expected to behave as an OTG host. This mode is specified based on the ID pin state alone, and does not guarantee that a device has been connected. The FX3 device can either enable the VBus at this stage or wait for the peripheral to initiate an SRP request.

CY_U3P_OTG_TYPE_B_CABLE OTG B-type peripheral cable connected to FX3. FX3 is expected to act as an OTG device by default. The FX3 device is expected to wait for the VBus to be valid. If this does not happen, then CyU3POtgSrpStart API can be invoked for initiating SRP.

CY_U3P_OTG_TYPE_ACA_A_CHG ACA RID_A_CHG charger. FX3 is expected to behave as OTG host. VBus is already available from the charger.

CY_U3P_OTG_TYPE_ACA_B_CHG ACA RID_B_CHG charger. FX3 can charge and initiate SRP. The remote host is not asserting VBus or is absent.

CY_U3P_OTG_TYPE_ACA_C_CHG ACA RID_C_CHG charger. FX3 device can charge and can connect but cannot initiate SRP as VBus is already asserted by the remote host.

CY_U3P_OTG_TYPE_MOT_MPX200 Motorola MPX.200 VPA

CY_U3P_OTG_TYPE_MOT_CHG Motorola non-intelligent charger.

CY_U3P_OTG_TYPE_MOT_MID Motorola mid rate charger.

CY_U3P_OTG_TYPE_MOT_FAST Motorola fast charger.

5.35.4 Function Documentation

5.35.4.1 CyU3POtgMode_t CyU3POtgGetMode (void)

Retrieve the currently selected OTG operating mode.

Description

This function retrieves the active OTG operation mode.

Return value

OTG mode selected at the time CyU3POtgStart call.

See Also

CyU3POtgMode_t
CyU3POtgStart

5.35.4.2 CyU3POtgPeripheralType_t CyU3POtgGetPeripheralType (void)

Identify the type of USB peripheral attached to FX3.

Description

This function returns the type of USB peripheral attached to FX3. This function can return the correct value only if there is a valid VBus or VBatt.

Return value

Type of peripheral attached (CyU3POtgPeripheralType_t)

See Also

CyU3POtgPeripheralType_t

5.35.4.3 CyU3PReturnStatus_t CyU3POtgHnpEnable (CyBool_t *isEnable*)

Initiate a HNP role change.

Description

This API initiates a OTG role change. This should only be called when both the host and device mode USB stacks in FX3 firmware are disabled.

If the isEnable parameter is true, the following sequence is performed:

If this API is called when in 'A' session, the API assumes that the remote device wants to get host role and will allow subsequent CyU3PUsbStart call to go through. If this API is called when in 'B' session, the API assumes that the remote host wants to relinqush control and will allow subsequent CyU3PUsbHostStart call to go through. It should be noted that the previous configuration has to be stopped before invoking this call.

If the isEnable parameter is false, the API allows the devices to revert to their original roles. The previous configuration has to be stopped before calling this API.

The following sequence should be followed when FX3 is default USB host:

1. The FX3 hosts sends down the SetFeature request for a_hnp_support indicating to the remote device that the host can do a role change. This is required for only legacy peripherals.
2. Remote devices request for an HNP via the session request bit.
3. Finish / abort all on-going transfers.
4. FX3 host sends down the SetFeature request for b_hnp_enable. This is to indicate to the remote device that the host is ready to initiate a role change.
5. The CyU3PUsbHostPortSuspend() API should be used to suspend the port.
6. Once the port is suspended, FX3 should wait for the remote device to get disconnected (CY_U3P_USB_HOST_EVENT_DISCONNECT host event). If this does not happen within the spec defined time, then the host can either resume host mode operation or it can end the session.
7. If the remote device got disconnected, FX3 should then call CyU3PUsbHostStop () to stop the host mode of operation.
8. Enable role change by calling CyU3POtgHnpEnable (CyTrue).
9. Call CyU3PUsbStart () to start the device mode stack.

The following sequence should be followed when FX3 is default USB device:

1. When FX3 requires a role change, it should first respond to a OTG GetStatus request with the session request flag set. When using fast enumeration mode, this can be done using CyU3POtgRequestHnp (CyTrue) call.
2. FX3 should wait until the remote host enables HNP by issuing SetFeature request for b_hnp_enable.
3. Once the SetFeature request is received, FX3 should wait for the bus to be suspended within the spec defined time. If this does not happen, then the device can either disconnect and end session or it can remain connected until the remote host resumes operation.
4. If the bus is suspended, then disconnect from the USB bus by calling CyU3PConnectState (CyFalse, CyFalse).
5. Now FX3 should stop the device mode stack by cleaning up all DMA channels and then finally calling CyU3PUsbStop ().
6. FX3 should then start the role change by calling CyU3POtgHnpEnable (CyTrue).
7. FX3 should then start the host mode operation by calling CyU3PUsbHostStart ().

Note

As HNP is role reversal, it has to be explicitly disabled. This only gets disabled if the user calls CyU3POtgStop() or CyU3POtgHnpEnable(CyFalse); or if the OTG peripheral is disconnected.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_SUPPORTED - Not in OTG mode.

CY_U3P_ERROR_NOT_STARTED - The module is not stated.

CY_U3P_ERROR_INVALID_SEQUENCE - Device or host stack is still active.

See Also

CyU3POtgStart
CyU3POtgRequestHnp
CyU3POtgStop
CyU3PUsbStart
CyU3PUsbStop
CyU3PUsbHostStart
CyU3PUsbHostStop
CyU3PUsbHostPortSuspend

Parameters

isEnable	Whether to initiate or reverse the HNP role change.

5.35.4.4 CyBool_t CyU3POtgIsDeviceMode (void)

Check whether USB device (peripheral) mode operation is permitted.

Description

This function determines the mode of OTG operation and checks whether the FX3 can initiate the device (peripheral) mode of operation.

Return value

CyTrue - Device mode of operation is allowed.

CyFalse - Device mode of operation is not allowed.

See Also

CyU3POtgStart
CyU3PUsbStart

5.35.4.5 CyBool_t CyU3POtgIsHnpEnabled (void)

Check if a role reversal mode is active or not.

Description

This API checks whether HNP role reversal is currently requested by the user.

Return value

CyTrue - HNP role change is active.

CyFalse - HNP role change is not active.

See Also

CyU3POtgStart
CyU3POtgEventCallback_t

5.35.4.6 CyBool_t CyU3POtgIsHostMode (void)

Check whether USB host mode operation is permitted.

Description

This function determines the mode of OTG operation and checks whether the FX3 can initiate host mode operation.

Return value

CyTrue - Host mode of operation is allowed.

CyFalse - Host mode of operation is not allowed.

See Also

CyU3POtgStart
CyU3PUsbHostStart

5.35.4.7 CyBool_t CyU3POtgIsStarted (void)

Check whether the OTG module has been started.

Description

This API can be used to check whether the OTG module in the FX3 firmware has been started.

Return value

CyTrue - OTG module has been started.

CyFalse - OTG module is not running.

See Also

CyU3POtgStart
CyU3POtgStop

5.35.4.8 CyBool_t CyU3POtgIsVBusValid (void)

Check if a valid VBus is available.

Description

The API can be used to determine the state of the VBus. Since the USB module can function properly only with the VBus enabled, this can be used to determine when to start the device / host stacks.

Notification of VBus state change can be received through the registered OTG event callback function as well.

Return value

CyTrue - VBus is valid.

CyFalse - A valid VBus is not available.

See Also

CyU3POtgStart
CyU3POtgEventCallback_t

5.35.4.9 CyU3PReturnStatus_t CyU3POtgRequestHnp (CyBool_t *isEnable*)

Set the HNP request bit in the device status word.

Description

This API is valid only in device mode of operation. To initiate HNP, the device need to set the session request bit in the device status word. If USB control requests are being handled in user callback, this needs to be handled by the user.

Since this is a role change request, the flag is not cleared until CyU3PUsbStop or CyU3POtgRequestHnp (CyFalse) is called, or there is an OTG peripheral change. In case of a change in the OTG peripheral attached, the library will automatically clear the session request flag.

Return value

CY_U3P_SUCCESS - The call was successful. CY_U3P_ERROR_NOT_SUPPORTED - Not in OTG mode. CY_U3P_ERROR_NOT_STARTED - Device stack is not started.

See Also

CyU3POtgStart
CyU3POtgHnpEnable

Parameters

isEnable	Whether to set or clear the host request flag.

5.35.4.10 CyU3PReturnStatus_t CyU3POtgSrpAbort (void)

Abort SRP request.

Description

The USB driver repeats the SRP request until VBus is detected. This function instructs the driver to abort the periodic SRP requests.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The module is not started.

CY_U3P_ERROR_INVALID_SEQUENCE - Wrong mode or host stack is still running.

See Also

CyU3POtgStart
CyU3POtgSrpStart

5.35.4.11 CyU3PReturnStatus_t CyU3POtgSrpStart (uint32_t *repeatInterval*)

Initiate an SRP request.

Description

This API is valid only when the FX3 is functioning as a USB 2.0 device, and is used to start the SRP request. This is expected to cause the USB host to enable the VBus voltage.

The CY_U3P_OTG_VBUS_VALID_CHANGE event is sent when a valid VBus voltage is detected. The OTG module will automatically repeat the SRP request until the VBus voltage is valid, or until CyU3POtgStpAbort() has been called.

The SRP repetition interval in milli-seconds is set through a parameter to this API.

Return value

CY_U3P_SUCCESS - The call was successful.

CY_U3P_ERROR_NOT_STARTED - The module is not yet started.

CY_U3P_ERROR_BAD_ARGUMENT - Input parameter is invalid.

CY_U3P_ERROR_INVALID_SEQUENCE - Host / device stack is still active or not in the correct mode.

See Also

CyU3POtgStart
CyU3POtgSrpAbort

Parameters

repeatInterval	SRP repeat interval in ms. The valid range is from 500 ms to 10s.

5.35.4.12 CyU3PReturnStatus_t CyU3POtgStart (CyU3POtgConfig_t * *cfg*)

Initialize the OTG module.

Description

The function initializes the OTG functionality in the FX3 USB block. At this point, the device is ready to detect any device/host/charger connection based on the state of the ID pin. Once the mode of operation is detected by the USB driver and notified through the OTG event callback, the corresponding start API needs to be invoked.

Carkit mode allows the USB 2.0 D+ and D- lines to be routed to either P-Port IOs or the UART TX/RX IO lines. These IOs cannot be used for the normal function when the carkit mode is active. Also, USB connections cannot be enabled when the carkit mode is active. The carkit mode can be disabled by invoking the CyU3POtgStop API.

Return value

CY_U3P_SUCCESS - when the configuration was successful.

CY_U3P_ERROR_NOT_SUPPORTED - if the FX3 device in use does not support the OTG feature.

CY_U3P_ERROR_NULL_POINTER - if the input parameter is NULL.

CY_U3P_ERROR_ALREADY_STARTED - the module was already started.

CY_U3P_ERROR_INVALID_SEQUENCE - the CY_U3P_OTG_MODE_DEVICE_ONLY mode was already started.

CY_U3P_ERROR_BAD_ARGUMENT - some input parameters are invalid.

See Also

CyU3POtgConfig_t
CyU3POtgStop
CyU3POtgIsStarted
CyU3PUsbStart
CyU3PUsbHostStart

Parameters

cfg	OTG configuration information.

5.35.4.13 CyU3PReturnStatus_t CyU3POtgStop (void)

Disable the OTG module.

Description

This function disables OTG mode detection on the FX3 device. This expects that the USB host/device mode operation has been shut down by calling the corresponding stop function.

Return value

CY_U3P_SUCCESS - The API call was successful.

CY_U3P_ERROR_NOT_STARTED - The module has not been started yet.

CY_U3P_ERROR_INVALID_SEQUENCE - The device or host stack is running.

See Also

CyU3PUsbStop
CyU3PUsbHostStop
CyU3POtgStart
CyU3POtgIsStarted

5.36 firmware/u3p_firmware/inc/cyu3utils.h File Reference

Utility functions provided by the FX3 library.

```
#include "cyu3types.h"
#include "assert.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Macros

- #define CY_U3P_MIN(a, b) (((a) > (b)) ? (b) : (a))
  Find the minimum of two numbers.
- #define CY_U3P_MAX(a, b) (((a) > (b)) ? (a) : (b))
  Find the maximum of two numbers.
- #define CY_U3P_MAKEWORD(u, l) ((uint16_t)(((u) << 8) | (l)))
  Create a word (16-bit) from two 8-bit numbers.
- #define CY_U3P_GET_LSB(w) ((uint8_t)((w) & UINT8_MAX))
  Get the LS byte from a 16-bit number.
- #define CY_U3P_GET_LSW(d) ((uint16_t)((d) & UINT16_MAX))
  Get the LS word from a 32-bit number.
- #define CY_U3P_GET_MSB(w) ((uint8_t)((w) >> 8))
  Get the MS byte from a 16-bit number.
- #define CY_U3P_GET_MSW(d) ((uint16_t)((d) >> 16))
  Get the MS word from a 32-bit number.
- #define CY_U3P_MAKEDWORD(b3, b2, b1, b0)
  Create a double word (32-bit) from four 8-bit numbers.
- #define CY_U3P_DWORD_GET_BYTE0(d) ((uint8_t)((d) & 0xFF))
  Retrieves byte 0 from a 32 bit number.
- #define CY_U3P_DWORD_GET_BYTE1(d) ((uint8_t)(((d) >> 8) & 0xFF))
  Retrieves byte 1 from a 32 bit number.
- #define CY_U3P_DWORD_GET_BYTE2(d) ((uint8_t)(((d) >> 16) & 0xFF))
  Retrieves byte 2 from a 32 bit number.
- #define CY_U3P_DWORD_GET_BYTE3(d) ((uint8_t)(((d) >> 24) & 0xFF))
  Retrieves byte 3 from a 32 bit number.
- #define CyU3PAssert(cond) assert(cond)
  Assert function used to log errors when expected conditions are not satisfied.

Functions

- void CyU3PMemCopy32 (uint32_t *dest, uint32_t *src, uint32_t count)

  Copy data 32-bits at a time from one memory location to another.
- void CyU3PBusyWait (uint16_t usWait)

  Delay function based on busy spinning in a loop.
- CyU3PReturnStatus_t CyU3PComputeChecksum (uint32_t *buffer, uint32_t length, uint32_t *chkSum)

  Compute a checksum over a user specified data buffer.
- CyU3PReturnStatus_t CyU3PReadDeviceRegisters (uvint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf)

  Function to read one or more FX3 device registers.
- CyU3PReturnStatus_t CyU3PWriteDeviceRegisters (uvint32_t *regAddr, uint8_t numRegs, uint32_t *dataBuf)

  Function to write one or more FX3 device registers.
- void **__aeabi_memset** (void *dest, size_t n, int c)
- void **__aeabi_memset4** (void *dest, size_t n, int c)
- void **__aeabi_memclr** (void *dest, size_t n)
- void **__aeabi_memclr4** (void *dest, size_t n)
- void **__aeabi_memcpy** (void *dest, const void *src, size_t n)
- void **__aeabi_memcpy4** (void *dest, const void *src, size_t n)

5.36.1 Detailed Description

Utility functions provided by the FX3 library. **Description**

The utility APIs are generic functions that are provided as helpers for the FX3 APIs.

5.36.2 Macro Definition Documentation

5.36.2.1 #define CY_U3P_MAKEDWORD(*b3, b2, b1, b0*)

Value:

```
((uint32_t)((((uint32_t)(b3)) << 24) | (((uint32_t)(b2)) << 16) | \
                                       (((uint32_t)(b1)) << 8) | ((uint32_t)(b0))))
```

Create a double word (32-bit) from four 8-bit numbers.

5.36.3 Function Documentation

5.36.3.1 void CyU3PBusyWait (uint16_t *usWait*)

Delay function based on busy spinning in a loop.

Description

This function is used to insert small delays (of the order of micro-seconds) into the firmware application. The delay is implemented using a busy spin loop and can be used anywhere. This API should not be used for large delays as other lower or same priority threads will not be able to run during this.

Return value

None

See Also

CyU3PThreadSleep

Parameters

usWait	Delay duration in micro-seconds.

5.36.3.2 CyU3PReturnStatus_t CyU3PComputeChecksum (uint32_t * *buffer*, uint32_t *length*, uint32_t * *chkSum*)

Compute a checksum over a user specified data buffer.

Description

This function computes the binary sum of all values in a user specified data buffer and can be used as a simple checksum to verify data consistency. This checksum API is used by the boot-loader to determine the checksum as well.

Return value

CY_U3P_SUCCESS if the checksum is successfully computed.

CY_U3P_ERROR_BAD_ARGUMENT if the parameters provided are erroneous.

Parameters

buffer	Pointer to data buffer on which to calculate the checksum.
length	Length of data buffer on which to calculate the checksum.
chkSum	Pointer to buffer that will be filled with the checksum.

5.36.3.3 void CyU3PMemCopy32 (uint32_t * *dest*, uint32_t * *src*, uint32_t *count*)

Copy data 32-bits at a time from one memory location to another.

Description

This is a memcpy equivalent function. This requires that the addresses provided are four byte aligned. Since the ARM core is 32-bit, this API does a faster copy of data than the 1 byte equivalent (CyU3PMemCopy). This API is also used by the firmware library. The implementation does not handle the case of overlapping buffers.

Return value

None

See Also

CyU3PMemCopy

Parameters

dest	Pointer to destination buffer.
src	Pointer to source buffer.
count	Size of the buffer (in words) to be copied.

5.36.3.4 CyU3PReturnStatus_t CyU3PReadDeviceRegisters (uvint32_t * *regAddr*, uint8_t *numRegs*, uint32_t * *dataBuf*)

Function to read one or more FX3 device registers.

Description

The FX3 device hardware implements a number of Control and Status registers that govern the behavior of and report the current status of each of the blocks. This function is used to read one or more contiguous registers from the register space of the FX3 device.

Return value

CY_U3P_SUCCESS if the register read(s) is/are successful.

CY_U3P_ERROR_BAD_ARGUMENT if the arguments passed in are erroneous.

See Also

CyU3PWriteDeviceRegisters

Parameters

regAddr	Address of first register to be read.
numRegs	Number of registers to be read.
dataBuf	Pointer to data buffer into which the registers are to be read.

5.36.3.5 CyU3PReturnStatus_t CyU3PWriteDeviceRegisters (uvint32_t * *regAddr*, uint8_t *numRegs*, uint32_t * *dataBuf*)

Function to write one or more FX3 device registers.

Description

This function is used to write one or more contiguous registers from the register space of the FX3 device.

Note

Use this function with caution and preferably under guidance from Cypress support personnel. The function does not implement any validity/side effect checks on the values being written into the registers.

Return value

CY_U3P_SUCCESS if the register write(s) is/are successful.

CY_U3P_ERROR_BAD_ARGUMENT if the arguments passed in are erroneous.

See Also

CyU3PReadDeviceRegisters

Parameters

regAddr	Address of first register to be written.
numRegs	Number of registers to be written.
dataBuf	Pointer to data buffer containing data to be written to the registers.

5.37 firmware/u3p_firmware/inc/cyu3vic.h File Reference

Internal functions for managing the VIC and interrupts on the FX3 device.

```
#include "cyu3types.h"
#include "cyu3externcstart.h"
#include "cyu3externcend.h"
```

Enumerations

- enum CyU3PVicVector_t { CY_U3P_VIC_GCTL_CORE_VECTOR = 0, CY_U3P_VIC_SWI_VECTOR, CY_U3P_VIC_DEBUG_RX_VECTOR, CY_U3P_VIC_DEBUG_TX_VECTOR, CY_U3P_VIC_WDT_VECTOR, CY_U3P_VIC_BIAS_CORRECT_VECTOR, CY_U3P_VIC_PIB_DMA_VE-

CTOR, CY_U3P_VIC_PIB_CORE_VECTOR, CY_U3P_VIC_UIB_DMA_VECTOR, CY_U3P_VIC_UIB_CORE_VECTOR, CY_U3P_VIC_UIB_CONTROL_VECTOR, CY_U3P_VIC_SIB_DMA_VECTOR, CY_U3P_VIC_SIB0_CORE_VECTOR, CY_U3P_VIC_SIB1_CORE_VECTOR, CY_U3P_VIC_RESERVED_15_VECTOR, CY_U3P_VIC_I2C_CORE_VECTOR, CY_U3P_VIC_I2S_CORE_VECTOR, CY_U3P_VIC_SPI_CORE_VECTOR, CY_U3P_VIC_UART_CORE_VECTOR, CY_U3P_VIC_GPIO_CORE_VECTOR, CY_U3P_VIC_LPP_DMA_VECTOR, CY_U3P_VIC_GCTL_PWR_VECTOR, CY_U3P_VIC_NUM_VECTORS }

Vector numbers for different interrupt sources.

Functions

- void CyU3PVicEnableInt (uint32_t vectorNum)

  Enable the interrupt for the specified vector number.
- void CyU3PVicDisableInt (uint32_t vectorNum)

  Disable the interrupt for the specified vector number.
- void CyU3PVicClearInt (void)

  Clear any interrupt that has occurred.
- uint32_t CyU3PVicIntGetStatus (void)

  Get the raw interrupt status.
- uint32_t CyU3PVicIRQGetStatus (void)

  Get the IRQ interrupt status.
- void CyU3PVicIntSetPriority (uint32_t vectorNum, uint32_t priority)

  Set the priority level of the interrupt vector.
- uint32_t CyU3PVicIntGetPriority (uint32_t vectorNum)

  Get the priority level of the interrupt vector.
- void CyU3PVicInit (void)

  The function initializes the VIC.
- void CyU3PVicSetupIntVectors (void)

  The function initializes the interrupt vector table.
- uint32_t CyU3PVicDisableAllInterrupts (void)

  This function disables all FX3 interrupts at the VIC level.
- void CyU3PVicEnableInterrupts (uint32_t mask)

  This function enables the specified interrupts at the VIC level.

5.37.1 Detailed Description

Internal functions for managing the VIC and interrupts on the FX3 device.

5.37.2 Enumeration Type Documentation

5.37.2.1 enum CyU3PVicVector_t

Vector numbers for different interrupt sources.

Description

The following enumeration lists the interrupt vector numbers on the FX3 device.

Enumerator

CY_U3P_VIC_GCTL_CORE_VECTOR 0: Low power mode entry/exit interrupt.

CY_U3P_VIC_SWI_VECTOR 1: Software interrupt.
CY_U3P_VIC_DEBUG_RX_VECTOR 2: Unused debug vector.
CY_U3P_VIC_DEBUG_TX_VECTOR 3: Unused debug vector.
CY_U3P_VIC_WDT_VECTOR 4: Watchdog timer interrupt.
CY_U3P_VIC_BIAS_CORRECT_VECTOR 5: Timer for PVT Bias correction.
CY_U3P_VIC_PIB_DMA_VECTOR 6: GPIF (PIB) DMA interrupt.
CY_U3P_VIC_PIB_CORE_VECTOR 7: GPIF (PIB) Core interrupt.
CY_U3P_VIC_UIB_DMA_VECTOR 8: USB DMA interrupt.
CY_U3P_VIC_UIB_CORE_VECTOR 9: USB core interrupt.
CY_U3P_VIC_UIB_CONTROL_VECTOR 10: Unused interrupt vector.
CY_U3P_VIC_SIB_DMA_VECTOR 11: Storage port DMA interrupt (FX3S only).
CY_U3P_VIC_SIB0_CORE_VECTOR 12: Storage port 0 core interrupt (FX3S only).
CY_U3P_VIC_SIB1_CORE_VECTOR 13: Storage port 1 core interrupt (FX3S only).
CY_U3P_VIC_RESERVED_15_VECTOR 14: Unused interrupt vector.
CY_U3P_VIC_I2C_CORE_VECTOR 15: I2C block interrupt.
CY_U3P_VIC_I2S_CORE_VECTOR 16: I2S block interrupt.
CY_U3P_VIC_SPI_CORE_VECTOR 17: SPI block interrupt.
CY_U3P_VIC_UART_CORE_VECTOR 18: UART block interrupt.
CY_U3P_VIC_GPIO_CORE_VECTOR 19: GPIO block interrupt.
CY_U3P_VIC_LPP_DMA_VECTOR 20: Serial peripheral (I2C, I2S, SPI and UART) DMA interrupt.
CY_U3P_VIC_GCTL_PWR_VECTOR 21: VBus detect interrupt.
CY_U3P_VIC_NUM_VECTORS Number of valid FX3 interrupt vectors.

5.37.3 Function Documentation

5.37.3.1 void CyU3PVicClearInt (void)

Clear any interrupt that has occurred.

Description

This function marks the current interrupt cleared at the VIC level, so that the VIC can raise the next pending interrupt. The actual cause of the interrupt has to be cleared or masked out before doing this.

This function should not be used directly by the user application.

Return value

None

See Also

CyU3PVicEnableInt
CyU3PVicDisableInt

5.37.3.2 uint32_t CyU3PVicDisableAllInterrupts (void)

This function disables all FX3 interrupts at the VIC level.

Description

This function can be used to disable all FX3 interrupts at the VIC level. The function returns a mask that represent the interrupts that were enabled before this function was called. It is expected that this mask would be used to re-enable the interrupts using the CyU3PVicEnableInterrupts function.

Return value

A mask that represents the interrupts that were enabled before this call.

See Also

CyU3PVicEnableInterrupts

5.37.3.3 void CyU3PVicDisableInt (uint32_t *vectorNum*)

Disable the interrupt for the specified vector number.

Description

Disable the interrupt with the specified vector number. All FX3 interrupts are enabled/disabled at appropriate times by the corresponding drivers in the FX3 firmware. This function should not be directly used by the user application.

Return value

None

See Also

CyU3PVicEnableInt
CyU3PVicClearInt

Parameters

vectorNum	Interrupt vector number (0 - CY_U3P_VIC_NUM_VECTORS) to be disabled.

5.37.3.4 void CyU3PVicEnableInt (uint32_t *vectorNum*)

Enable the interrupt for the specified vector number.

Description

Enable the interrupt with the specified vector number. All FX3 interrupts are enabled/disabled at appropriate times by the corresponding drivers in the FX3 firmware. This function should not be directly used by the user application.

Return value

None

See Also

CyU3PVicDisableInt
CyU3PVicClearInt

Parameters

vectorNum	Vector number to be enabled (0 - CY_U3P_VIC_NUM_VECTORS).

5.37.3.5 void CyU3PVicEnableInterrupts (uint32_t *mask*)

This function enables the specified interrupts at the VIC level.

Description

This function can be used to re-enable interrupts that were previously disabled through the CyU3PVicDisableAll-Interrupts function. These two functions can be used together to save and restore interrupt states across critical sections of code.

Return value

None

See Also

CyU3PVicDisableAllInterrupts

Parameters

mask	Bit-mask representing interrupts to be enabled.

5.37.3.6 void CyU3PVicInit (void)

The function initializes the VIC.

Description

This function initializes the PL192 Vectored Interrupt Controller on the FX3 device and sets up the interrupt vector addresses for all FX3 device interrupts.

This function is called internally during device initialization and should not be called directly.

Return value

None

5.37.3.7 uint32_t CyU3PVicIntGetPriority (uint32_t *vectorNum*)

Get the priority level of the interrupt vector.

Description

Returns the currently programmed priority level for the specified interrupt vector. 0 is the highest priority level, and 15 is the lowest priority level.

Return value

The priority level for the specified interrupt vector.

See Also

CyU3PVicIntSetPriority

Parameters

vectorNum	Interrupt vector number.

5.37.3.8 uint32_t CyU3PVicIntGetStatus (void)

Get the raw interrupt status.

Description

Get a bit vector that represents the current status of all FX3 device interrupts. The bit vector will report an interrupt even if it is currently disabled at the VIC level.

Return value

Bit vector reporting the current status of all FX3 interrupts.

See Also

CyU3PVicIRQGetStatus

5.37.3.9 void CyU3PVicIntSetPriority (uint32_t *vectorNum,* uint32_t *priority*)

Set the priority level of the interrupt vector.

Description

Set the priority level for a specified interrupt vector. This function is not used in the library and is not expected to be called. Interrupts in FX3 firmware are split into two groups: a high priority group which is not pre-emptable; and a low priority group that is pre-emptable.

Return value

None

See Also

CyU3PVicIntGetPriority

Parameters

vectorNum	Interrupt vector number (0 - CY_U3P_VIC_NUM_VECTORS).
priority	Priority level to be set (0 - 15)

5.37.3.10 uint32_t CyU3PVicIRQGetStatus (void)

Get the IRQ interrupt status.

Description

This function returns a bit vector that reports the interrupt vectors that are both active and enabled.

Return value

Bit vector representing the active and enabled interrupts.

See Also

CyU3PVicIntGetStatus

5.37.3.11 void CyU3PVicSetupIntVectors (void)

The function initializes the interrupt vector table.

Description

This function sets up the interrupt vector address table for the FX3 device, and is called internally by the library during device initialization.

Return value

None

See Also

CyU3PVicInit

Index