AT43USB370
Software Development Guide

Revision 0.9

Contents

List of Tables

List of Figures

About this Document

This document describes the AT43USB370 system software library and the associated Application Protocol Interfaces (API). It provides examples to illustrate how to develop application using the AT43USB370 APIs.

Intended Audience

This document is written for the software developers to aid the development of applications for the AT43USB370. It assumes that readers are familiar with the AT43USB370 architecture.

Using this document

This document is organized into following chapters.

Chapter 1	INTRODUCTION TO SOFTWARE DEVELOPMENT GUIDE
Chapter 2	The AT43USB370 Library
Chapter 3	High Level Host Application Programming Interface
Chapter 4	Device Application Programming Interface
Chapter 5	Low Level Host Application Programming Interface
Chapter 6	Low Level Device Application Programming Interface
Chapter 7	System Processor Interface Register Set

Typographical conventions

The following typographical conventions are used in this document

`Courier`	Denotes the text such as program and file names, function names, source code etc
`Courier italic`	Denotes arguments to functions where the argument would be replaced with a specific value
`Courier bold`	Denotes language key words, function return types, register names

1 INTRODUCTION TO SOFTWARE DEVELOPMENT GUIDE

1.1 Overview

The AT43USB370 system software library is a collection of USB libraries and associated Application Protocol Interfaces (APIs) designed to hide the complexities of the USB development form system and software designers. . It enables rapid development of USB device drivers and applications for the AT43USB370 based products.

The AT43USB system API set encapsulates the complete USB functionality . It is comprised of a high level API set and a register-level API set that together, provide complete and flexible access to the AT43USB370 USB functionality. Extensive examples of API usage are given in this document to minimize the learning curve.

1.2 Features

- AT43USB370 Embedded USB Firmware
  - RTOS Agnostic
  - Autonomous USB Functionality Handling in both Host and Function Modes
  - Complete USB Hub Driver
  - Support for up to 15 downstream USB devices concurrently
- AT43USB370 System Software Library
  - ANSI C Compliant, high level and register level APIs
  - Modular, Target System and RTOS Independent Architecture
  - Function Mode Architecture Supporting Multiple Configurations, Interfaces, Endpoints and Alternate Settings
  - Configurable Interface Through Software Interrupts for Reporting Responses and Connection / Disconnection Events
  - Supports for all USB Data Transfer Types (Control, Bulk, Interrupt, Isochronous)
  - Single API Function Call to Execute Standard USB Requests
  - Re-entrant, thread safe API function calls

1.3 Limitations

The AT43USB370 can support a maximum of 15 downstream USB devices concurrently.

1.4 AT43USB370 Software Architecture

The AT43USB370 operates in one of two modes – host mode or function mode. The mode of operation is determined by the firmware running on the USB Controller (USBC) and System Interface Controller (SIC) of the AT43USB370. The operating mode also dictates the required AT43USB370 system software library resident on the system processor. The following sections describe the firmware architecture of the AT43USB370 in the host and function (or device) mode.

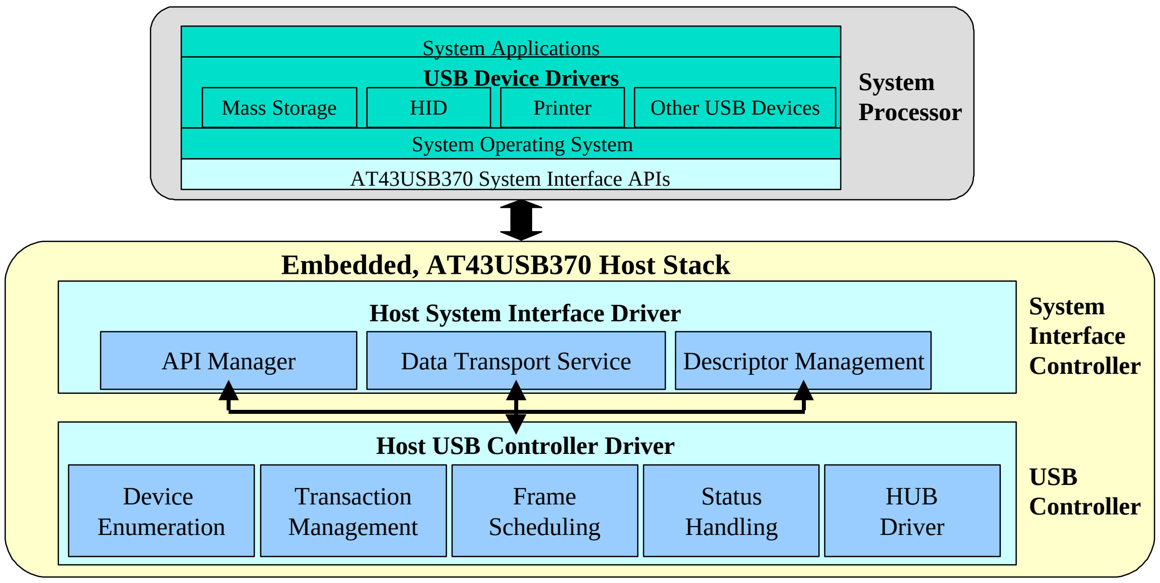
1.4.1 Host Software Architecture

The host software architecture is comprised of a set of USB firmware embedded in the AT43UB370 and a set of AT43USB370 specific, USB host system library resident on the system processor.

The embedded USB firmware consists of the Host USB Controller Driver (HUSBCD) and System Interface Controller Driver (HSICD). The HUSBCD and HSICD are collectively referred to as the Embedded USB host stack. The HUSBCD implements hardware dependent USB protocols whereas the HSICD manages the data flow between the system processor and the AT43USB370.
The USB host system library serves as the link between the AT43USB370 host stack and the applications running on the system processor. It interfaces with the system application though a set of AT43USB370 APIs. This API set encapsulates the complete USB host functionality and is used to develop USB device drivers and applications of any type.

Figure 1 illustrates the major functional blocks comprising the AT43USB370's host software model. In a typically usage scenario, one or more USB device drivers and system application can access the AT43USB370 USB host functionality through the AT43USB370 APIs.

Figure 1 AT43USB370 Host Firmware Architecture



1.4.1.1 Host USB Controller Driver

The HUSBCD runs on the USBC when the AT43USB370 operates as a USB Host. This driver interacts with the AT43USB370 hardware and performs USB protocol management relating to the USB host operation. More specifically, this driver performs the following tasks.

– **Hub Driver**
The HUSBCD embeds a complete Hub Class driver to provide autonomous Hub support.

– **Device Enumeration**
The HUSBCD handles all the connection / disconnection related events, including full enumeration of a newly connected device (either on the root port or on a hub).

– **Transaction Management**

The HUSBCD automatically schedules transactions using the information it receives from the devices during enumeration. Isochronous and Interrupt transactions are given up to a maximum of 90% of the frame time. Bulk and Control transfers are guaranteed the remaining 10% and any of the time not being consumed by the Isochronous and Interrupt transfers.

– **Frame Scheduling**
Frame management involves calculating the time required for the next transaction and transaction completion prediction as described in USB 2.0 Specification. It also includes the calculations, the Host Controller has to make at the time of enumeration to ensure that the requirements of the newly connected device can be met with in the current bandwidth budget of the Host and whether or not a particular (non-isochronous) transfer can be executed in a given frame.

– **Status Handling**
After a transaction is completed, the HUSBCD posts the transaction status to the SIC.

1.4.1.2 System Interface Controller Driver

The HSICD runs on the SIC when the AT43USB370 operates as a USB Host. This driver performs the following tasks.

– **Data Transport Services**
The HSICD handles the data transfer between the AT43USB370 (FIFOs) and the external system processor's memory.

– **API Manager**
The HSICD manages all the information exchange with the external system processor. Besides taking commands from the external system processor and reporting back their status, this also includes reporting any new device / bus related events to the external system processor.

– **Descriptor Management**
The HSICD gathers the USB descriptors from the devices and reports back to system processor through API function calls.

1.4.1.3 USB Host System Library

The USB host system library is comprised of the AT43USB370 host system interface APIs and the underlying software library. The USB host system library is resident on the system processor and interacts with the system application through the AT43USB370 APIs. The API set encapsulates the complete USB functionality. It is ANSI C compliant and serves as the building blocks of USB device drivers of any type. System application can also interface with the APIs directly without going through the USB device drivers.

1.4.1.4 System Software

System software runs on the external system processor. It comprises of the optional USB device drivers and the system applications.

– **USB Device Drivers**
This layer contains the standard or application specific USB device drivers built on top of the AT43USB370's API.

– **System Application**
  The system application communicates to the AT43USB370 host either through the AT43USB370 Host System Interface APIs directly or through the standard or application specific USB device drivers built on top the AT43USB370

The AT43USB370 APIs can be integrated with the system software using the Library Integration functions as described in Chapter 2. Please refer to Chapter 3 for detailed descriptions of the host API set.

1.4.2 Device Software Architecture

Similar to the host software model, the device software architecture is comprised of a set of USB firmware embedded in the AT43UB370 and a set of AT43USB370 specific, USB device system library resident on the system processor.

Figure 2 shows the major blocks comprising the AT43USB370's Device software model.

The embedded USB device firmware consists of the Device USB Controller Driver (DUSBCD) and System Interface Controller Driver (DSICD The DUSBCD and DSICD are collectively referred to as the Embedded USB Device Stack. The DUSBCD implements hardware dependent USB protocols whereas the DSICD manages the data flow between the system processor and the AT43USB370.

The USB device system library serves as the link between the AT43SUB370 embedded firmware and the system software. It interfaces with the system software through a set of AT43USB370 Device System Interface APIs. This API set encapsulates the complete USB device functionality and allows the system software to dynamically configure the AT43USB370 to support a variety of USB device configuration (number of endpoints, type of endpoints, type of transfers, etc.).

The system software is divide into the Device Function Driver and the Device Application. In terms of the USB command and data flow, all USB protocol requests issued by the USB host are handled by the DUSBCD. Standard USB device class requests and proprietary device driver specific requests are passed to the DSICD which in turns passes these requests to the Device Function Driver. The device function driver, in conjunction with the device application software, processes the requests and responds back to the host. In addition, requests involving data transfer to and from AT43USB370 endpoints of the device are passed to the Device Function Driver.

1.4.2.1 Device USB Controller Driver

The DUSBCD runs on the USBC when the AT43USB370 operates as a USB Device. This driver interacts with the AT43USB370 hardware and performs USB protocol management relating to the USB device operation. In particular, the DUSBCD performs the following functions.

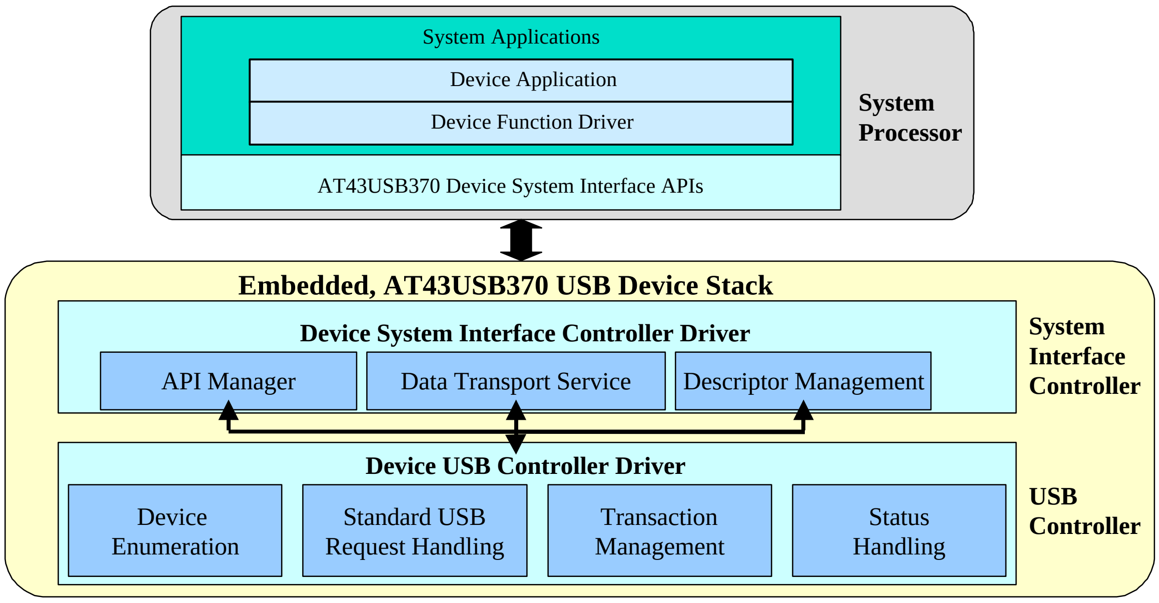
– **Device Enumeration**
  The DUSBCD generates the connect status and handles the enumeration process of the AT43USB370 device when it is connected to the USB host.

– **USB Protocol Request Handling**
  It responds to all standard USB protocol layer requests issued by the USB host autonomously.

– **Transaction Handling**
The DUSBCD automatically processes incoming packets from the OUT endpoints for isochronous, bulk and interrupt transactions. These transactions are subsequently reported to the Device Function Driver. Similarly, the DUSBCD receives data from the Device Function Driver and passes the data upstream to the USB Host via IN endpoints.

– **Status Handling**
The DUSBCD posts the transaction status to the DSICD, once a transaction is completed.

Figure 2 AT43USB370 Device Firmware Architecture


System Applications
Device Application
Device Function Driver
AT43USB370 Device System Interface APIs
System Processor
Embedded, AT43USB370 USB Device Stack
Device System Interface Controller Driver
API Manager
Data Transport Service
Descriptor Management
System Interface Controller
Device USB Controller Driver
Device Enumeration
Standard USB Request Handling
Transaction Management
Status Handling
USB Controller

1.4.2.2 Device System Interface Controller Driver

The DSICD runs on the System Interface Controller. This driver is responsible for the following functions.

– **Data Transport Services**
The DSICD handles the data transfer between the AT43USB370 device and the system processor's memory. It also provides an interface for the system processor to specify endpoint configurations and the requisite USB descriptor table.

– **API Manager**
The DSICD manages all information (commands and status) exchange with the external system. It reports status events to the Device Function Driver which uses the event information to keep track of the device activities at any given time.

– **Descriptor Management**

The DSICD gathers the USB descriptors from the devices and reports back to system processor through API function calls.

1.4.2.3 USB Device System Interface Library APIs

The USB device system interface library is comprised of the AT43USB370 device system interface APIs and the underlying software library. The USB device system library is resident on the system processor and interacts with the system application through the APIs. The API set encapsulates the complete USB functionality. It is ANSI C compliant and serves as the building blocks of USB device applications of any type.

1.4.2.4 System Application

The System application runs on the system processor. It comprises of the Device Function Driver and the Device Application.

– **Device Function Driver**
  The device function driver is responsible for handling standard USB class or proprietary USB device driver requests from the USB host, in addition to handling USB data transfer.

– **Device Application**
  This is the system application running on the system processor.

Device function drivers and device applications are very application specific. As such, they are not part of the standard AT43USB370 USB device firmware library.

The system application interacts with the AT43USB370 device through AT43USB370's Device APIs. These APIs can be integrated with the system software using the Library Integration functions as described in the Chapter 2. Please refer to Chapter 4 for detailed descriptions of the Device APIs.

1.5 Terminology

Table 1 Terminology

AT43USB370	USB Embedded Host/Function Processor
Command	System software initiated activity. Function call from the system software to the AT43USB370
ISR	Interrupt Service Routine
Load buffer	Memory buffer allocated by the system for AT43USB370 to read from. See Buffer requirements in Section 1.4
Response	Function call from AT43USB370 to the system software on SWI
Request	AT43USB370 initiated activity.
Store buffer	Memory buffer allocated by the system for AT43USB370 to write into. See Buffer requirements in Section 1.4
SWI	Software Interrupt

System Processor	The processor to which the AT43USB370 is connected as a peripheral
System software	The application software running on the system microprocessor for AT43USB370 host/device
USBPlib	AT43USB370's USB Processor Library or the AT43USB370's system library
DUSBCD	Device USB Controller Driver
HUSBCD	Host USB Controller Driver
HSIC	Host System Interface Controller
DSIC	Device System Interface Controller
HSICD	Host System Interface Controller Driver
DSICD	Device System Interface Controller Driver
HUSBC	Host USB Controller
DUSBC	Device USB Controller

Note: The term Function and Device are use interchangeably throughout the entire document.

2 *The AT43USB370 Library*

2.1 About the USBP Library

The USBP library is the AT43USB370's host and device system library. The primary functions of the USBP library are:

- Initializes the AT43USB370 internal controllers
- Provides an communication and control link between the embedded AT43USB370 firmware and the system application
- Interacts with the system application in both host and function mode through the AT43USB370 system interface APIs.

2.2 API Organization

The USBP Library comprises of a High Level API set and a Low (Register) Level API set. The register level API set makes use of the System Processor Interface Register Set of the AT43USB370 to interact with the AT43USB370. The high-level API set is built upon the Register Level API set. It provides another layer of abstraction that hides the AT43USB370's system processor interface and its internal operations from software developers.

Each of the high and low level API is divided into two main components

– **Host APIs**
  This API set contains the APIs specific to the AT43USB370's host mode. Chapter 3 and Chapter 5 provide more details of the high level and register level Host APIs respectively.

– **Device APIs**
  This API set contains APIs specific to the AT43USB370 function mode. Chapter 4 and Chapter 6 provide more details of the high level and register level Device API respectively.

The USBP library is ANSI C compliant and can be easily ported on different system processors and ROTS.

Atmel has taken great care to create the high level APIs that can be used to construct virtually any type of USB device drivers and application. It is highly recommended that developers base all USB drivers and applications on the high level APIs exclusively. The register level APIs should NOT be used unless absolutely necessary.

2.2.1 High Level API

The high level API set is organized into the following three types of functions.

2.2.1.1 Command Functions

These functions are called by the application to issue commands to the AT43USB370. These functions either return an immediate response, or returns CMD_IN_PROGRESS to indicate that the application have to call the related response function to get the final response.

2.2.1.2 Response Functions

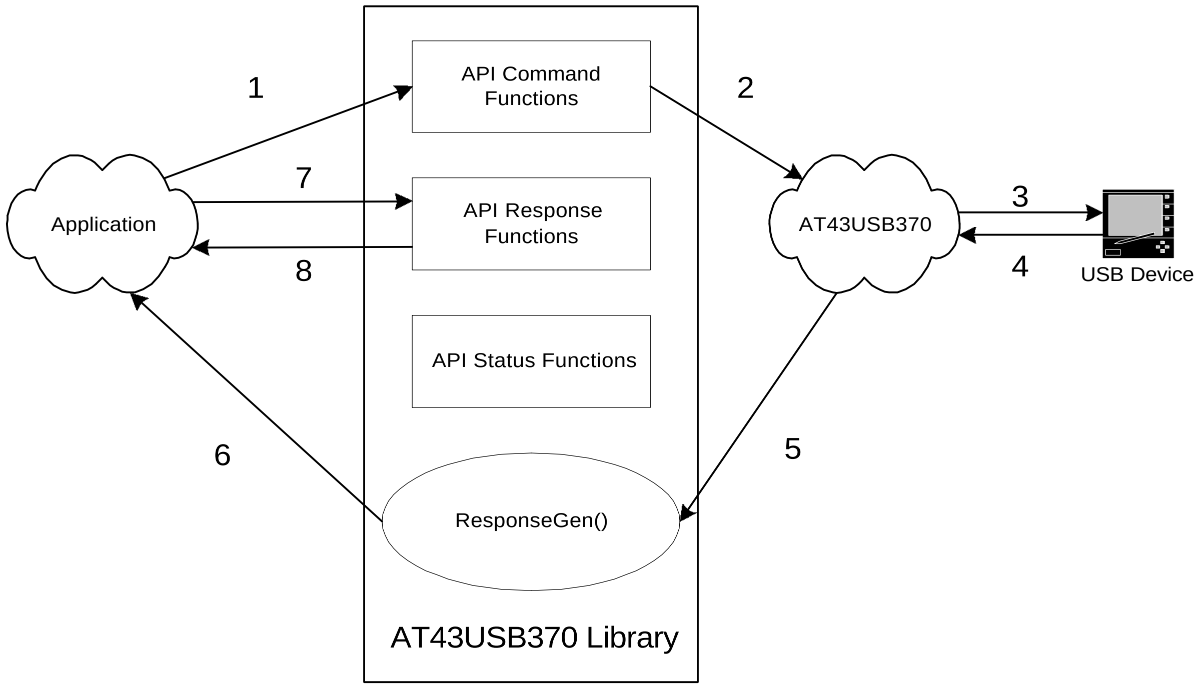
These functions are called by the application to get the final response of the command issued to the AT43USB370.

2.2.1.3 Status Functions

These functions provide the current status or status change information to the application (i.e. device connection, and device disconnection).

2.2.1.4 Sequence of Command / Response Functions

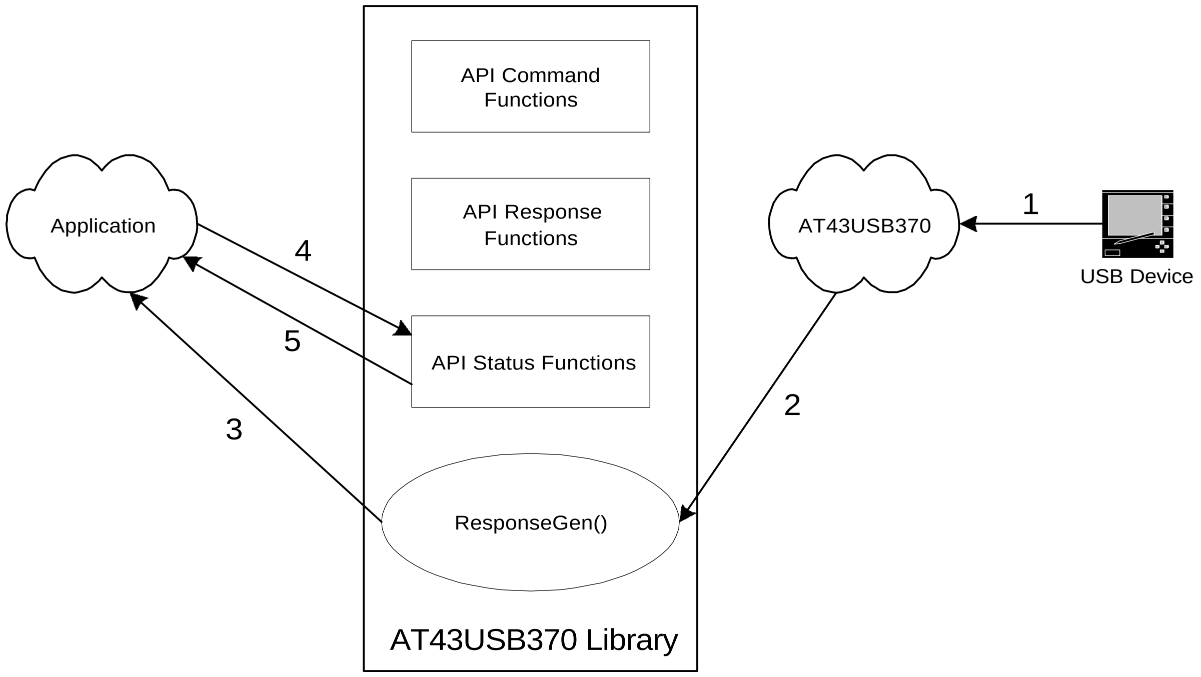
Figure 3: Sequence of Command / Response Functions



1. Application calls the API command function to issue command to the AT43USB370 Library .
2. The AT43USB370 Library requests the AT43USB370 to schedule the transfer.
3. The AT43USB370 generates USB traffic according to the transfer scheduled.
4. Data is transferred between the AT43USB370 and a USB device or host. When the data transfer is completed, the response is returned by the AT43USB370 to the AT43USB370 library .
5. The AT43USB370 library calls the `USBPlib_ResponseGen()` function with appropriate argument.
6. `USBPlib_ResponseGen()` function call by the AT43USB370 library indicates to the application that the final response of the command has arrived. The argument of this function determines the kind of response.
7. Application calls the corresponding API Response function to get the final response.
8. The API Response function returns the final response to the application.

2.2.1.5 Sequence of Status Functions Calls

Figure 4: Sequence of Status Functions Calls.



1. A status event is detected by the AT43USB70. The device connection and disconnection events are treated as the status events.
2. The status event is reported to the AT43USB370 library.
3. The AT43USB370 library calls the `USBPlib_ResponseGen()` function with appropriate argument. This indicates to the application that a status event has occurred.
4. Application calls appropriate status function to get the details of the event.
5. The API Status function returns the status information to the application.

2.2.2 Low Level API

The low level API set consists of Commands and Requests. The High Level API Commands are issued by the system software to the AT43USB370 to initiate transaction(s) and bus activities. The Requests are issued by the AT43USB370 to the system software to give responses or report status events. Chapter 5 and Chapter 6 describe the Command and Request Sets implemented for the AT43USB370 host and device (function) mode respectively.

2.3 Adding USBP Library to System Software

To use USBP library in a project, AT43USB370.h and AT43USB370Lib_xxx.a must be included in the project. Details of the AT43USB370Lib_xxx.a library file are provided in the release note of the library

.

2.3.1 System Software Integration Functions

The following functions are required for library integration into the system software:

- `USBPlib_Init()`
- `USBPlib_ResponseGen()`

2.3.1.1 USBPlib_Init()

This function initializes the USBP Library and the AT43USB370 firmware. This function must be called once at the start of the application before any API call is made.

Arguments
`void`

Return Value
`void`

Example
Assuming that main() is the entry point of the application

```
int main(void)
{
   //Call platform specific initialization routines
   //Call any application specific initialization routines

   USBPlib_init();  //initializing USBP library and AT43USB370 firmware

   //Enable interrupts
   // Any AT43USB370 API call may now be made
}
```

2.3.1.2 USBPlib_ResponseGen()

An API call may return intermediate response structure that indicates that the final response will be issued at a later time (`CMD_IN_PROGRESS` in the `CmdStatus` field). For all of such API calls, `USBPlib_ResponseGen()`is called by the application to issue the final response.

Argument
`unsigned int ResponseNumber`

Table 2 describes the complete list of arguments for `ResponseNumber`. Arguments not described in this table are reserved. For the details of the corresponding functions, please see the sections Host API and Device API.

Table 2 USBPlib_ResponseGen () Arguments

USBPlib_ResponseGen() Argument	**Argument**	**Status/Response Function Name**
H_FINAL_CMD_RESPONSE	*0x110*	GetFinalResponse
H_DEV_CONNECTED	*0x111*	DeviceConnected
H_DEV_DISCONNECTED	*0x112*	DeviceDisconnected
D_RESET	*0x501*	N.A
D_SUSPENDED	*0x502*	N.A
D_RESUMED	*0x503*	N.A
D_DISCONNECTED	*0x504*	N.A
D_SET_CONFIGURATION	*0x508*	DevSetConfiguration
D_SET_INTERFACE	*0x509*	DevSetInterface
D_SET_FEATURE	*0x50A*	DevSetFeature
D_CLEAR_FEATURE	*0x50B*	DevClearFeature
D_FINAL_CMD_RESPONSE	*0X50C*	DevFinalCmdResponse
D_OUT_DATA_RCVD	*0x50E*	DevOutDataRcvd

All the `USBPlib_ResponseGen()` arguments are defined in AT43USB370.h

Return Value
`void`

Example

The `USBPlib_ResponseGen()`is implemented as part of a software interrupt handler. The Interrupt Service Routine (ISR) passes ResponseNumber as an argument to the USBPlib_ResponseGen(). USBPlib_ResponseGen() parses the response number and calls the associated status functions which returns the response. A typical example of the USBPlib_ResponseGen() is shown below:

```
void USBPlib_ResponseGen (unsigned int ResponseNumber)
{
   if (ResponseNumber == H_FINAL_CMD_RESPONSE)
   {
      sFinalCmdResponse svFinalCmdResponse;
      svFinalCmdResponse = GetFinalResponse();              // Response API
   }
   else if (ResponseNumber == H_DEV_CONNECTED)
   {
      sDevConnect sConnectedDeviceInfo;                     // Status API

      sConnectedDeviceInfo = DeviceConnected();
   }
   else if (ResponseNumber == H_DEV_DISCONNECTED)
   {
      unsigned char ucDeviceDisconnected;
      ucDeviceDisconnected = DeviceDisconnected();          // Status API
   }
}
```

3 *High Level Host Application Programming Interface*

3.1 AT43USB370 Host Mode

The AT43USB370 can operate as a USB host. An USB host implementation requires generic driver device support for a variety of USB devices. The AT43USB370's high level host API is designed specifically to simplify the development and the subsequent integration of USB device drivers into system applications. System software is typically comprised of a USB device driver layer and the application layer. It runs on the system processor and communicates with the downstream USB device(s) through the AT43USB370 host system library and the associated high level APIs.

The high level host API functions are grouped into the following three categories.

- ***Host Status Functions***
  Report connection and disconnection events to the system software.

- ***Host Command Functions***
  Send USB-specific and data transfer requests to the devices connected to the AT43USB370.

- ***Host Response Functions***
  Report the responses of the command functions after they are executed by AT43USB370 host.

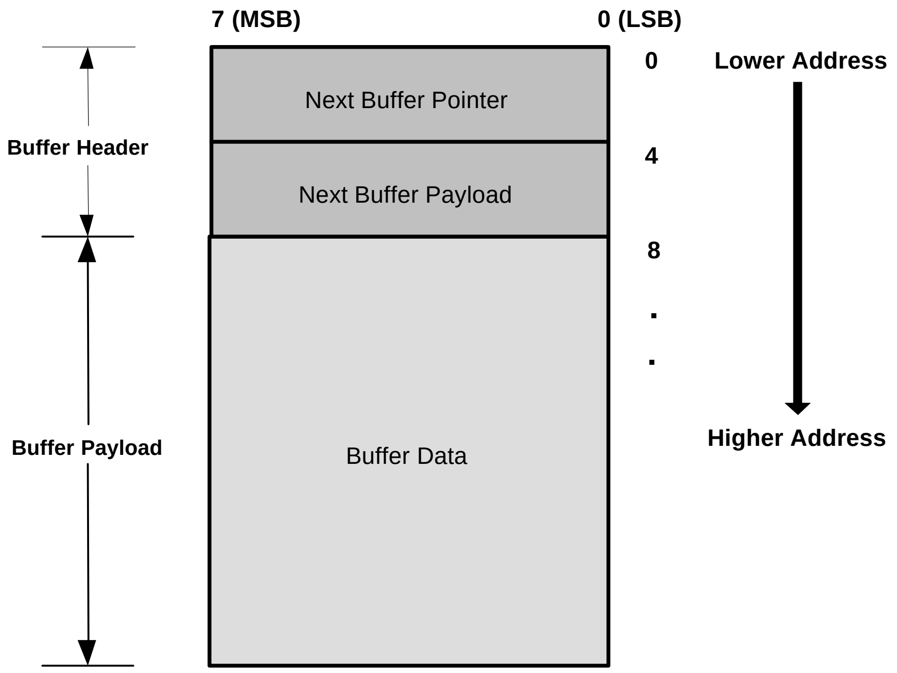
3.2 Buffer Structure

The data is exchanged between the application and the AT43USB370 library through data buffers provided by the system software. All the API functions that involve data transfer between the application and AT43USB370's FIFOs require a pointer to the data buffer for incoming or out going data. The data buffer may be allocated from the system processor's memory while calling the API command function and freed when the final response is returned by the AT43USB370 through API response function.

The AT43USB370 library requires the system software to provide a single buffer or multiple linked buffers. A single buffer may be sufficient for executing small transactions whereas multiple buffers may be required where a large contiguous memory is not available in the system processor memory space to execute large transactions. In that case, various small buffers available may be linked to form a buffer linked list and provided to the AT43USB370 library. The AT43USB370 facilitates data transfer to and from these buffers.

For a single buffer, alignment is not required at the end of the buffer. For multiple buffers, the end of every buffer except the last one must be aligned at 4-byte boundary. At the start of each buffer there is an 8-byte header that contains the pointer and payload of the next buffer. The start of every buffer is required to be aligned at 4 byte boundary. Figure 5 shows the required format of a buffer:

Figure 5 General Buffer Structure



"Next Buffer Payload" contains the size of the next buffer in the buffer linked list. This value excludes the 8-byte buffer header.

"Next Buffer Pointer" is a pointer to the start of the next buffer in the buffer linked list.

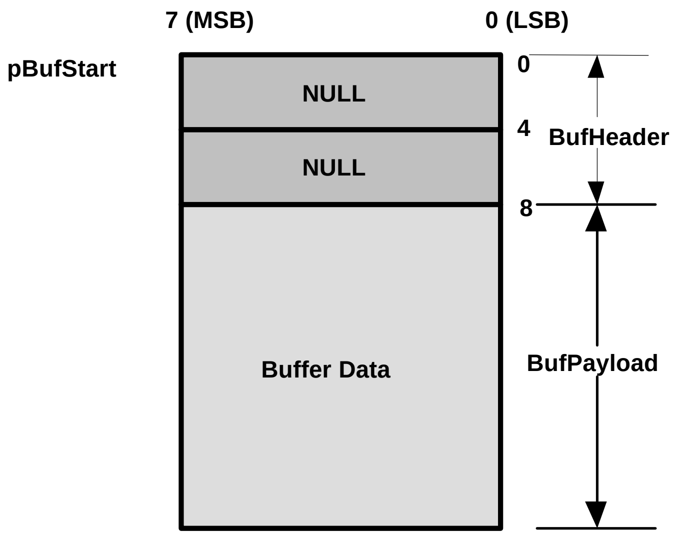
3.2.1 Single Buffer Structure

To receive 18-byte device descriptor (per USB specification) from an USB device, a single buffer is allocated from the system memory. The following example illustrates the initialization of a receiving buffer.

```
unsigned char BufHeader;
unsigned int  *pBufStart;
unsigned int  BufPayload;

BufHeader = 8; // 8-Byte buffer header
BufPayload = 18; // Example value

pBufStart = (unsigned int *) malloc (BufPayload + BufHeader);
*(pBufStart) = 0; // Setting Next Buffer pointer  to null
*(pBufStart + 1 ) = 0; // Setting Next Buffer payload to null
```

Figure 6shows how a single buffer would look like after allocation from the system memory and its initialization.

Figure 6 Single Buffer Structure



The ***Next Buffer Payload*** and ***Next Buffer Pointer*** fields of the single buffer must always be set to null (0x0). Alignment is not required for single buffer.

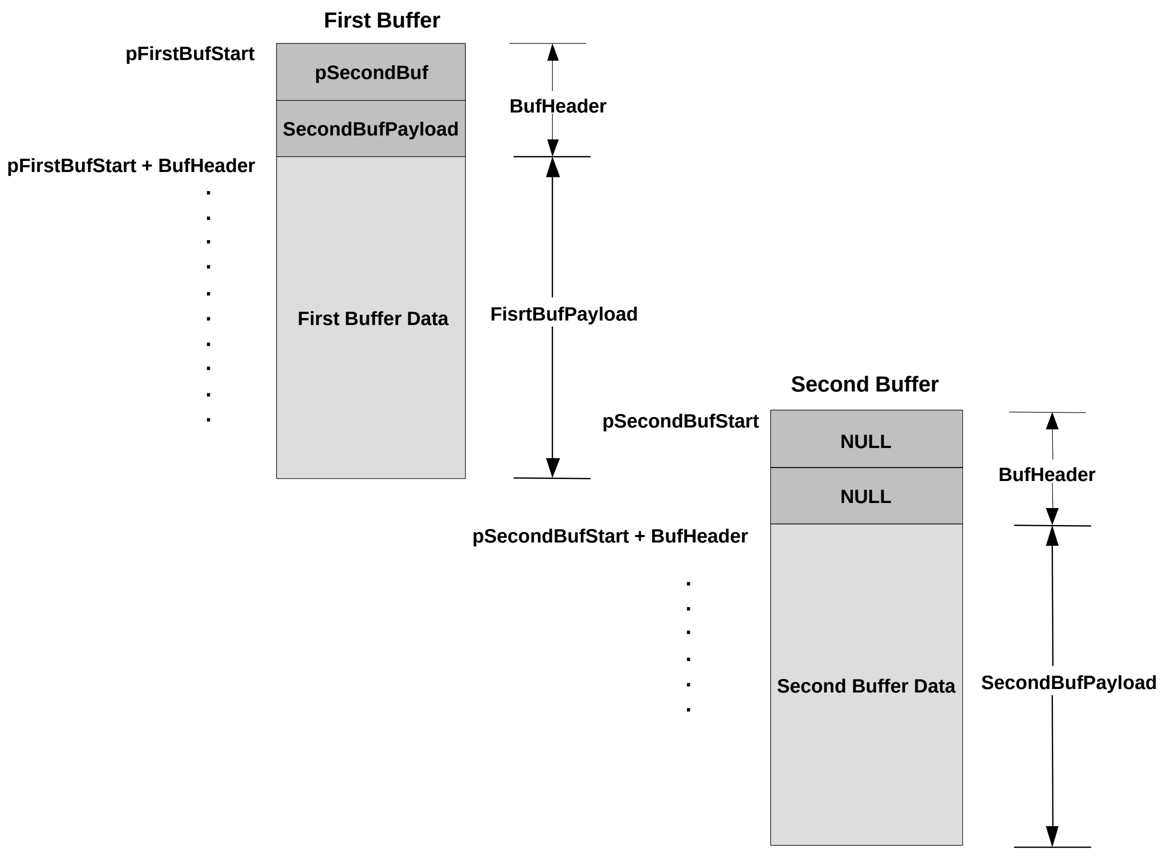
3.2.2 Multiple Buffer Structure

Multiple buffers are allocated from the system memory and initialized as follows:

```
unsigned char BufHeader;
unsigned int *pFirstBufStart;
unsigned int FirstBufPayload;
unsigned int *pSecondBufStart;
unsigned int SecondBufPayload;

BufHeader = 8; // 8-Byte buffer header
FirstBufPayload = 12; // Example value
SecondBufPayload = 20; // Example value

pFistBufStart = (unsigned int *) malloc (FirstBufPayload + BufHeader);
pSecondBufStart = (unsigned int *) malloc (SecondBufPayload + BufHeader);

*(pFirstBufStart) = (unsigned int ) pSecondBufStart; // Setting the next
                                                       // buffer pointer

*(pBufStart + 1 ) = SecondBufPayload; // Setting next buffer payload

*(pSecondBufStart) = 0x0;      // Setting Next Buffer pointer
*(pSecondBufStart + 1 ) = 0x0; // Setting Next Buffer payload
```

Figure 7 shows how multiple buffers would look like after allocation from the system memory and their initialization.

Figure 7 Multiple Buffers Structure



The ***Next Buffer Payload*** and ***Next Buffer Pointer*** fields of the last buffer in the multiple buffer linked list must always be set to null (0x0) and the end of every buffer except the last one must be aligned at 4-byte boundary.

3.3 High Level Host API

3.3.1 Host Header File

The AT43USB370 provides a header file (**AT43USB370.h**) to the system software through which various configuration constants of the AT43USB370 may be set. These constants are described below:

- **AT43USB370_MODE**
- **USBPLib**

The AT43USB370 is capable of operating as a USB host or a USB function (device). The system software can configure AT43USB370 to act in one of these modes by setting **AT43USB370_MODE** to a value specified in the following table.

AT43USB370_MODE	**Value**
HOST_MODE	0x1
DEVICE_MODE	0x2

To configure the AT43USB370 as USB Host, this constant (AT43USB370_MODE) is set to 0x1 as shown below:

```
#define AT43USB370_MODE                 HOST_MODE
```

This definition governs the AT43USB370's mode of operation at startup. During run-time, the mode can be switched to the other by downloading the appropriate firmware into the AT43USB370 and resetting it.

3.3.2 Host Status Functions

The host status functions report the connection and disconnection events to the system software. When any such event is detected by the AT43USB370, API function `USBPlib_ResponseGen()` is called by the AT3USB370 library to inform the system software about the status event. The argument to this function specifies the event for which the status is generated by the AT43USB370. Depending upon the argument, the system software calls the respective status API function which returns the status information.

Table 3 Host Status Functions

Return Type	Status Function Name	Argument	USBPlib_ResponseGen() Argurment
`SDevConnect`	`DeviceConnected`	*`void`*	`H_DEV_CONNECTED`
`unsigned char`	`DeviceDisconnected`	*`void`*	`H_DEV_DISCONNECTED`

The following sections describe these status functions.

3.3.2.1 DeviceConnected ()

The DeviceConnected() function is used to report the device connection event to the system software. When a USB device is connected to AT43USB370's root port or through a hub, the AT43USB370 detects the connection, enumerates the device and assigns an address to it.

The AT43USB370, then, informs the system software about the device connection through `USBPlib_ResponseGen()`function with `H_DEV_CONNECTED (`defined in `AT43USB370.h)),` passed as argument to the function. The system software recognizes the event by seeing this argument and calls the host API status function `DeviceConnected()` which returns the device connection structure.

Syntax

`sDevConnect` `DeviceConnected`**`(void)`**

Return Value

Returns the structure of **sDevConnect** type as described below :

```
struct sDevConnect
{
   unsigned char  DevAddr;
   unsigned char  DevClass;
   unsigned char  DevSubClas;
   unsigned char  HubAddr;
   unsigned char  PortNum;
   unsigned short BCDRelNum;
   unsigned short VendorID;
   unsigned short ProductID;
};
```

where

DevAddr	is the device address of the device connected. This address is assigned by the AT43USB370 to the newly connected device. The system software uses this address in communicating with the device.
DevClass	is the class of the device. Depending upon the device functionality, this fields contains the **bDeviceClass** value of the device descriptor if it is non-zero. *Or* **bInterfaceClass** of the first interface descriptor of the device.
DevSubClass	is the SubClass of the device. Depending upon the device functionality, this fields contains the **bDeviceSubClass** value of the device descriptor if it is non-zero. *Or* **bInterfaceSubClass** of the first interface descriptor of the device.
HubAddr	is the device address of the hub to which the device is connected. If the device is connected to the root port, this field contains 0x0.
PortNum	is the port number of the hub to which the device is connected. If the device is connected to the root port, this field contains 0x0.
BCDRelNum	is the **bcdUSB** value of the device descriptor.
VendorID	is the **VendorID** value of the device descriptor.
ProductID	is the **ProductID** value of the device descriptor.

The AT43USB370 provides the above information about the newly connected device to let the software properly identify the device and load appropriate driver for it.

3.3.2.2 DeviceDisconnected ()

This status functions is used to report the device disconnection event to the system software. When a USB device is disconnected from the AT43USB370, the AT43USB370 detects the disconnection and informs the system software about the device disconnection through `USBPlib_ResponseGen()`function with `H_DEV_DISCONNECTED` (defined in `AT43USB370.h`), passed as argument to the function. The system software recognizes the event by seeing this argument and calls the Host API status function `DeviceDisconnected()` which returns the device address of the device disconnected.

Syntax

`unsigned char` `DeviceDisconnected`**`(void)`**

Return value

Returns the device address of the device disconnected.

3.3.3 Host Command Functions

The AT43USB370 provides the following set of host command functions for interacting with the AT43USB370.

Table 4 Host Command Functions

Return Type	Function Name	Arguments
sIntCmdResponse	ResetDevice	*unsigned char DevAddr*
sIntCmdResponse	SuspendDevice	*unsigned char DevAddr*
sIntCmdResponse	ResumeDevice	*unsigned char DevAddr*
sIntCmdResponse	GetDeviceDescriptor	*unsigned char DevAddr, unsigned int *pBufStart, unsigned int BufPayload*
sIntCmdResponse	GetConfigDescriptor	*unsigned char DevAddr, unsigned char ConfigNumber, unsigned int *pBufStart, unsigned int BufPayload*
sIntCmdResponse	SetConfiguration	*unsigned char DevAddr, unsigned char ConfigNumber*
sIntCmdResponse	SetInterface	*unsigned char DevAddr, unsigned char InterfaceNumber, unsigned char AltSettingNumber*
sIntCmdResponse	GetIsoData	*unsigned char DevAddr, unsigned char EndpointAddr, unsigned short IsoPacketSize, unsigned int *pBufStart, unsigned int BufPayload*
sIntCmdResponse	SendIsoData	*unsigned char DevAddr, unsigned char EndpointAddr, unsigned short IsoPacketSize, unsigned int *pBufStart, unsigned int BufPayload*
sIntCmdResponse	GetData	*unsigned char DevAddr, unsigned char EndpointAddr, unsigned char RetryCount, unsigned char NAKCount, unsigned int *pBufStart,*

		unsigned int BufPayload
sIntCmdResponse	SendData	*unsigned char DevAddr,* *unsigned char EndpointAddr,* *unsigned char RetryCount,* *unsigned char NAKCount,* *unsigned int *pBufStart,* *unsigned int BufPayload*
sIntCmdResponse	AbortTransfer	*unsigned char CmdId*
sIntCmdResponse	ControlTransfer	*unsigned char DevAddr,* *unsigned char EndpointAddr,* *unsigned int SetupHi,* *unsigned int SetupLo,* *unsigned char DataStage,* *unsigned int *pBufStart,* *unsigned int BufPayload*
sIntCmdResponse	CustomTransfer	*unsigned char DevAddr,* *unsigned char EndpointAddr,* *unsigned char PacketType,* *unsigned char DataToggle,* *unsigned char RetryCount,* *unsigned char NAKCount,* *unsigned short IsoPacketSize,* *unsigned int *pBufStart,* *unsigned char BufPayload*
sIntCmdResponse	SetPortFeature	*unsigned char HubAddr,* *unsigned char PortNum,* *unsigned char FeatureSelector*
sIntCmdResponse	ClearPortFeature	*unsigned char HubAddr,* *unsigned char PortNum,* *unsigned char FeatureSelector*

3.3.3.1 Return Type – Host Command Functions

All Host API command functions return the structure of **sIntCmdResponse** type as described below.

```
struct sIntCmdResponse
{
   unsigned char CmdID;
   unsigned char CmdStatus;
};
```

where :

CmdID	is the Command ID of this command.
CmdStatus	is the intermediate command status. One of the following values is returned:

CmdStatus	Value
CMD_IN_PROGRESS	0x0

BANDWIDTH_NOT_AVAILABLE	0x1
POWER_NOT_AVAILABLE	0x2
CMD_ERROR	0x3
CMD_SUCC_EXE	0xF

`CmdStatus` values have the following meanings:

- *CMD_IN_PROGRESS*
Command is in progress. Final response would be delivered through the host response function `GetFinalResponse()` (Called by the system software in response to `USBPlib_ResponseGen()` function call by the AT43USB370 library with an argument H_FINAL_CMD_RESPONSE).

- *BANDWIDTH_NOT_AVAILABLE*
In response to `SetConfiguration`() and `SetInterface()`, if the AT43USB370 does not find enough bandwidth available on the bus for the selected configuration/alternate setting, it returns with this status in the intermediate response structure.

- *POWER_NOT_AVAILABLE*
In response to `SetConfiguration`(), if AT43USB370 does not find enough power available on the bus for the selected configuration, it returns with this status in the intermediate response structure.

- *CMD_ERROR*
Command Error has occurred. Invalid parameters specified while issuing the command.

- *CMD_SUCC_EXE*
Command has been successfully executed.

Note:
The system software can preserve the `CmdId` to retrieve the final command response received through the host response function. The system software must only expect the final response in case it is issued **CMD_IN_PROGRESS** in the `CmdStatus` field of this structure. For all other returned values, the system software treats the response as final from the AT43USB370.

3.3.3.2 ResetDevice ()

This function resets the specified device.

Syntax

`sIntCmdResponse` `ResetDevice (`**`unsigned char`** *`DevAddr`*`)`

where:

`DevAddr`	is the device address of the device to be reset.

Return Value

Returns the structure of **`sIntCmdResponse`** type.

Example

To reset the device with device address 0x6, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;

DevAddr = 0x6;  // Device Address of the device to be reset
svIntCmdResponse = ResetDevice (DevAddr);
```

3.3.3.3 SuspendDevice ()

This function suspends the specified device.

Syntax

`sIntCmdResponse` `SuspendDevice (`**`unsigned char`** *`DevAddr`*`)`

where:

DevAddr	is the device address of the device to be suspended.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To suspend the device with the device address 0xC, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;

DevAddr = 0xC;  // Device Address of the Device to be suspended
svIntCmdResponse = SuspendDevice (DevAddr);
```

3.3.3.4 ResumeDevice ()

This function resumes the specified device.

Syntax

`sIntCmdResponse` `ResumeDevice (`**`unsigned char`** *`DevAddr`*`)`
where:

DevAddr	is the device address of the device to be resumed.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To resume the device with the device address 0xC, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;

DevAddr = 0xC;  // Device Address of the Device to be resumed
svIntCmdResponse = ResumeDevice (DevAddr);
```

3.3.3.5 GetDeviceDescriptor ()

This function provides the device descriptor of the specified device. The system software provides the buffer to store the descriptor.

Syntax

```
sIntCmdResponse GetDeviceDescriptor (unsigned char DevAddr,
                                     unsigned int  *pBufStart,
                                     unsigned int  BufPayload)
```

where:

DevAddr	is the device address of the device whose descriptor is required.
pBufStart	is the pointer to the start of buffer provided by the system software for storing the descriptor.
BufPayload	is the length of the buffer provided for storing the descriptor. This value excludes the buffer header bytes. If the descriptor length exceeds the *BufPayload*, the AT43USB370 reports buffer overrun (RESP_BUF_OVERRUN) through the host response function.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To get the device descriptor of from the device with device address 0x5, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;
unsigned char BufHeader;
unsigned int *pBufStart;
unsigned int BufPayload;

BufHeader = 0x8; // 8-byte buffer header
DevAddr = 0x5; // Device Address of the target device.
BufPayload = 0x12; // Size of the device descriptor in bytes.
pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// storing the descriptor.  Additional 8 bytes are allocated to make the
// buffer header according to the AT43USB370 Buffer Requirement.  As a
single
// buffer is being allocated, the buffer header fields are initialized to
// zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = GetDeviceDescriptor (DevAddr, pBufStart, BufPayload);

// The descriptor will be stored in the buffer from location *(pBufStart +
// 2) onwards
```

3.3.3.6 GetConfigDescriptor ()

This function provides the configuration descriptor of a particular configuration of the device. The system software provides the buffer to store the descriptor.

Syntax

```
sIntCmdResponse GetConfigDescriptor (unsigned char DevAddr,
                                     unsigned char ConfigNumber,
                                     unsigned int  *pBufStart,
                                     unsigned int  BufPayload)
```

where:

DevAddr	is the device address of the device whose descriptor is required.
CongigNumber	is the configuration number of the configuration whose descriptor is requested.
pBufStart	is the pointer to the start of buffer provided by the system software for storing the descriptor.
BufPayload	is the length of the buffer provided for storing the descriptor. This value excludes the buffer header bytes. If the descriptor length exceeds the *BufPayload*, the AT43USB370 reports buffer overrun (RESP_BUF_OVERRUN) through the host response function.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To get the configuration descriptor of 0x8 bytes, configuration number 0x1 of a device with device address 0x15, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char ConfigNumber;
unsigned int *pBufStart;
unsigned int BufPayload;
```

```
DevAddr = 0x15; // Device Address of the target device.
BufHeader = 0x8; // 8-byte buffer header
ConfigNumber = 0x1; // Configuration Number.
BufPayload = 0x8; // Requesting first 8 bytes of the configuration
// descriptor.  The actual length of the descriptor will be known from these
// bytes and requested later.

pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// storing the descriptor.  Additional 8 bytes are allocated to make the
// buffer header according to the AT43USB370 Buffer Requirement.  As a
single
// buffer is being allocated, the buffer header fields are initialized to
// zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = GetConfigDescriptor (DevAddr, ConfigNumber, pBufStart,
                                        BufPayload);

// The descriptor will be stored in the buffer from location *(pBufStart +
// 2) onwards
```

3.3.3.7 SetConfiguration ()

This function sets a particular configuration on the device. The AT43USB370 does the resource management for the said configuration and if power and bandwidth are not available on the bus to support the desired configuration, the system software is informed of this through the intermediate response structure returned.

Syntax

```
sIntCmdResponse SetConfiguration (unsigned char DevAddr,
                                  unsigned char ConfigNumber)
```

where:

DevAddr	is the device address of the target device.
ConfigNumber	is the configuration number of the configuration to be set. This is the **bConfigurationValue** value of the specified configuration descriptor of the device.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To set the configuration number 0x1 on a device with the device address 0x10, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;
unsigned char ConfigNumber;

DevAddr = 0x10; // Device Address of the target device.
ConfigNumber = 0x1; // Configuration value of the configuration to set.

svIntCmdResponse = SetConfiguration (DevAddr, ConfigNumber);
```

3.3.3.8 SetInterface ()

This function sets a particular alternate setting of an interface of the device. A configuration must have already been set on the device using the `SetConfiguration ()` API command function. The AT43USB370 evaluates the bandwidth required to support the said alternate setting and if not available, the system software is informed of this through the intermediate response structure returned.

Syntax

```
sIntCmdResponse SetInterface (unsigned char DevAddr,
                              unsigned char InterfaceNumber,
                              unsigned char AltSettingNumber)
```

where:

DevAddr	is the device address of the target device.
InterfaceNumber	is the interface number of the interface whose alternate setting is to be set. This is the **bInterfaceNumber** value of the specified interface descriptor of the device.
AltSettingNumber	is the alternate setting number of the alternate setting to set. This is the **bAlternateSetting** value of the specified interface descriptor of the device.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To set the alternate setting 0x1 of interface 0x0, on a device with the device address 0x10, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char DevAddr;
unsigned char InterfaceNumber;
unsigned char AltSettingNumber;

DevAddr = 0x10; // Device Address of the target device.
InterfaceNumber = 0x0; // Interface value of the selected interface.
AltSettingNumber = 0x1; // Alternate Setting value to set.

svIntCmdResponse = SetInterface(DevAddr, InterfaceNumber,
                              AltSettingNumber);
```

3.3.3.9 GetIsoData ()

This function gets data from Isochronous endpoints of the device.

Syntax

```
sIntCmdResponse GetIsoData (unsigned char  DevAddr,
                            unsigned char  EndpointAddr,
                            unsigned short IsoPacketSize,
                            unsigned int   *pBufStart,
                            unsigned int   BufPayload)
```

where:

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target isochronous IN endpoint.
IsoPacketSize	is the packet size for the transfers associated with this command. This value is specified appropriately for rate matching the source and sink, while issuing the transaction to the Isochronous endpoints.
pBufStart	is the pointer to the start of buffer provided by the system software for storing the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To get isochronous data from an isochronous endpoint (endpoint address 0x4) of the device (device address 0x10), the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned short IsoPacketSize;
unsigned int *pBufStart;
unsigned int BufPayload;

BufHeader = 0x8; // 8-byte buffer header
DevAddr = 0x10; // Device Address of the target device.
EndpointAddr = 0x4; // Endpoint Address of the target endpoint.
IsoPacketSize = 0xB0; // 176 bytes per packet.
BufPayload = 704; // Requesting 4 packets of 176 bytes each.
pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// storing the data.  Additional 8 bytes are allocated to make the buffer
// header according to the AT43USB370 Buffer Requirement.  As a single
buffer
// is being allocated, the buffer header fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload
```

```
svIntCmdResponse = GetIsoData (DevAddr, EndpointAddr, IsoPacketSize,
                               pBufStart, BufPayload);

// The data returned from the Isochronous endpoint of the device will be
// stored in the buffer from location *(pBufStart + 2) onwards.
```

3.3.3.10 SendIsoData ()

This function sends data to isochronous endpoints of the device.

Syntax

```
sIntCmdResponse SendIsoData (unsigned char  DevAddr,
                             unsigned char  EndpointAddr,
                             unsigned short IsoPacketSize,
                             unsigned int   *pBufStart,
                             unsigned int   BufPayload)
```

where:

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target isochronous OUT endpoint.
IsoPacketSize	is the packet size for the transfers associated with this command. This value is specified appropriately for rate matching the source and sink, while issuing the transaction to the Isochronous endpoints.
pBufStart	is the pointer to the start of buffer provided by the system software for sending the data.
BufPayload	Is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To send isochronous data to an isochronous endpoint (endpoint address 0x4) of the device (device address 0x10), the function call would be::

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned short IsoPacketSize;
unsigned int *pBufStart;
unsigned int BufPayload;

unsigned char BufHeader;
DevAddr = 0x10; // Device Address of the target device.
EndpointAddr = 0x4; // Endpoint Address of the target endpoint.
IsoPacketSize = 0xB0; // 176 bytes per packet.
BufPayload = 704; // Sending 4 packets of 176 bytes each.
pBufStart = malloc (BufPayload + Bufheader); // Allocating memory for
```

```
// sending the data.  Additional 8 bytes are allocated to make the buffer
// header according to the AT43USB370 Buffer Requirement.  As a single
buffer
// is being allocated, the buffer header fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

// The data to be sent to the Isochronous endpoint of the device is //
stored by the system software in the buffer from location
// *(pBufStart + 2) onwards.

svIntCmdResponse = SendIsoData (DevAddr, EndpointAddr, IsoPacketSize,
                                pBufStart, BufPayload);
```

3.3.3.11 GetData ()

This function gets data from Non-Isochronous endpoints (Control, Bulk and Interrupt) of the device.

Syntax

```
sIntCmdResponse GetData (unsigned char DevAddr,
                         unsigned char EndpointAddr,
                         unsigned char RetryCount,
                         unsigned char NAKCount,
                         unsigned int  *pBufStart,
                         unsigned int  BufPayload)
```

where:

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target endpoint.
RetryCount	is the numbers of times, retries are allowed for the transactions associated with the command.
NAKCount	is the number of times the NAKs are allowed for the transactions associated with the command.
pBufStart	is the pointer to the start of store buffer provided by the system software for storing the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To get 1K (1024 bytes) data from bulk endpoint (endpoint address 0x4) of the device (device address 0x10), the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned int *pBufStart;
unsigned int BufPayload;

BufHeader = 0x8;
DevAddr = 0x10; // Device Address of the target device.
EndpointAddr = 0x4; // Endpoint Address of the target endpoint.
BufPayload = 1024; // Requesting 1K data from the IN  endpoint of the
// device.
pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// storing the data.  Additional 8 bytes are allocated to make the buffer
// header according to the AT43USB370 Buffer Requirement.  As a single
buffer
// is being allocated, the buffer header fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = GetData (DevAddr, EndpointAddr, pBufStart, BufPayload);

// The data returned from the addressed endpoint of the device will be
// stored in the buffer from location *(pBufStart + 2) onwards.
```

3.3.3.12 SendData ()

This function sends data to Non-Isochronous endpoints (Control, Bulk and Interrupt) of the device.

Syntax

```
sIntCmdResponse SendData(unsigned char DevAddr,
                         unsigned char EndpointAddr,
                         unsigned char RetryCount,
                         unsigned char NAKCount,
                         unsigned int  *pBufStart,
                         unsigned int  BufPayload)
```

where:

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target endpoint.
RetryCount	is the numbers of times, retries are allowed for the transactions associated with the command.
NAKCount	is the number of times the NAKs are allowed for the transactions associated with the command.
pBufStart	is the pointer to the start of store buffer provided by the system software for sending the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To send 1K (1024 bytes) data to bulk endpoint (endpoint address 0x3) of the device (device address 0x5), the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned char RetryCount;
unsigned char NAKCount;
unsigned int *pBufStart;
unsigned int BufPayload;

BufHeader = 0x8;
DevAddr = 0x5; // Device Address of the target device.
RetryCount = 10;   // Retry 10 times
NAKCount = 15      //20 NAK allowed
EndpointAddr = 0x3; // Endpoint Address of the target endpoint.
BufPayload = 1024; // Sending 1K Data.
pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// sending the data.  Additional 8 bytes are allocated to make the buffer
// header according to the AT43USB370 Buffer Requirement.  As a single
buffer
// is being allocated, the buffer header fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

// The data to be sent to the specified endpoint of the device is
// stored in the buffer from location *(pBufStart + 2) onwards.

svIntCmdResponse = SendData (DevAddr, EndpointAddr, RetryCount, NAKCount,
pBufStart, BufPayload);
```

3.3.3.13 AbortTransfer ()

This function aborts the transaction(s) associated with a particular Command ID. The function takes the command ID (`CmdId`), issued earlier by the AT43USB370 in the intermediate response structure, in return to any command function call, as an argument.

Syntax

sIntCmdResponse `AbortTransfer` (**unsigned char** *CmdId*)

where:

CmdID	is the Command ID of a particular command. This is the `CmdID` value of the `sIntCmdResponse` response structure returned while issuing the command through host command function.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To abort the transaction with Command ID 0xA, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char CmdID;

CmdID = 0xA; // Command ID of the command to be terminated.
svIntCmdResponse = AbortTransfer (CmdID);
```

3.3.3.14 ControlTransfer ()

This function performs transfers to the control endpoint of the device.

Syntax

```
sIntCmdResponse ControlTransfer (unsigned char DevAddr
                                 unsigned char EndpointAddr,
                                 unsigned int  SetupHi,
                                 unsigned int  SetupLo,
                                 unsigned char DataStage,
                                 unsigned int  *pBufStart,
                                 unsigned int  BufPayload)
```

where

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target endpoint.
SetupHi	are the least significant 4 bytes of the 8-byte setup data.
SetupLo	are the most significant 4 bytes of the 8-byte setup data.
DataStage	specifies whether a data stage will follow the setup stage and its direction.

The following table shows the definitions:

DataStage	Value	Description
DATA_STAGE_NULL	0x0	Setup stage will be followed by the status stage.
DATA_STAGE_IN	0x1	Data stage following the setup stage will be in the IN direction. The data will be received from the device.
DATA_STAGE_OUT	0x2	Data stage following the setup stage will be in the OUT direction. The data will be sent to the device.

pBufStart is the pointer to the start of store buffer provided by the system software for data transfer.

BufPayload is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To get the current device configuration value of a device (device address 0x5), the USB standard device request of Get Configuration can be sent through this function.

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned int  SetupHi;
unsigned int  SetupLo;
unsigned char DataStage;
unsigned int  *pBufStart;
unsigned int  BufPayload;

DevAddr = 0x5;
EndpointAddr = 0x0;
SetupHi = 0x00000880; // First 4 bytes of the setup data
SetupLo = 0x01000000; // Last 4 bytes of the setup data
DataStage = DATA_STAGE_IN; // IN Data stage in the control transaction.
BufHeader = 0x8; // 8-byte buffer header
BufPayload = 0x1; // 1 Byte will be returned by the device during the //
data stage of the transaction
pBufStart = malloc (BufPayload + BufHeader);
*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = ControlTransfer (DevAddr, EndpointAddr, SetupHi,
                                    SetupLo, DataStage, pBufStart,
                                    BufPayload);
```

One byte data is returned by the device during the data stage of control transaction indicating the current configuration value.

3.3.3.15 CustomTransfer ()

This functions performs custom transaction(s).

Syntax

```
sIntCmdResponse CustomTransfer (unsigned char DevAddr,
                                unsigned char EndpointAddr,
                                unsigned char PacketType,
                                unsigned char DataToggle,
                                unsigned char RetryCount,
                                unsigned char NAKCount,
                                unsigned int  IsoPacketSize,
                                unsigned int  *pBufStart,
                                unsigned int  BufPayload)
```

where:

DevAddr	is the device address of the target device.
EndpointAddr	is the endpoint address of the target endpoint.
PacketType	See the following definitions

PacketType	Value
PACKET_OUT	0x0
PACKET_IN	0x1
PACKET_SETUP	0x2

DataToggle See the following definitions

DataToggleValue	Value
DATA_TOGGLE_0	0x0
DATA_TOGGLE_1	0x1

RetryCount	is the numbers of times, retries are allowed for the transactions associated with the command.
NAKCount	is the number of times the NAKs are allowed for the transactions associated with the command.
IsoPacketSize	is the value appropriate for rate matching the source and sink, while issuing transaction to / from Isochronous endpoints.
pBufStart	is the pointer to the start of Load / Store buffer provided by the system software.
BufPayload	is the length of the buffer, excluding the buffer header, provided by the system software.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To perform custom transactions to a bulk out endpoint (endpoint address 0x4) of the device (device address 0x10) with data toggle 0x0, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char DevAddr;
unsigned char EndpointAddr;
unsigned char PacketType;
unsigned char DataToggle;
unsigned char RetryCount;
unsigned char NAKCount;
unsigned int  IsoPacketSize;
unsigned int  *pBufStart;
unsigned int  BufPayload;

BufHeader = 0x8;
DevAddr = 0x10;
EndpointAddr = 0x4;
PacketType = PACKET_OUT;
DataToggle = DATA_TOGGLE_0;
RetryCount = 0x3;
NAKCount = 0x3;
IsoPacketSize 0x0;
BufPayload = 1024;
pBufStart = malloc (BufPayload + BufHeader); // Allocating memory for
// sending the data.  Additional 8 bytes are allocated to make the buffer
// header according to the AT43USB370 Buffer Requirement.  As a single
buffer
// is being allocated, the buffer header fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = CustomTransfer (DevAddr, EndpointAddr, PacketType,
                                   DataToggle, RetryCount, NAKCount,
                                   IsoPacketSize, pBufStart,
                                   BufPayload);
```

3.3.3.16 SetPortFeature ()

This function sets or enables a particular feature on a hub port connected to the AT43USB370.

NOTE:
This function must only be called for Hub Class devices.

Syntax

```
sIntCmdResponse SetPortFeature (unsigned char HubAddr,
                                unsigned char PortNum,
                                unsigned char FeatureSelector)
```

where

HubAddr	is the device address of the target hub.
PortNum	is the number of port whose feature is to be set.
FeatureSelector	is the USB-Specific Hub Class Feature Selector value. Table 5 specifies these values.

Table 5 Hub Class Feature Selector values

FeatureSelector	Value
PORT_CONNECTION	0
PORT_ENABLE	1
PORT_SUSPEND	2

PORT_OVER_CURRENT	3
PORT_RESET	4
PORT_POWER	8
PORT_LOW_SPEED	9
C_PORT_CONNECTION	16
C_PORT_ENABLE	17
C_PORT_SUSPEND	18
C_PORT_OVER_CURRENT	19
C_PORT_RESET	20
PORT_TEST	21
PORT_INDICATOR	22

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To set the PORT_POWER feature on a port (port number 0x2) of the Hub (Hub Address 0x1), the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char HubAddr;
unsigned char PortNum;
unsigned char FeatureSelector;

HubAddr = 0x1;
PortNum = 0x2;
FeatureSelector = PORT_POWER; // Feature to be set.

svIntCmdResponse = SetPortFeature(HubAddr, PortNum, FeatureSelector);
```

3.3.3.17 ClearPortFeature ()

This function clears or disables a particular feature on a hub port connected to AT43USB370.

NOTE:
This function must only be called for Hub Class devices.

Syntax

```
sIntCmdResponse ClearPortFeature (unsigned char HubAddr,
                                  unsigned char PortNum,
                                  unsigned char FeatureSelector)
```

where

`HubAddr`	is the device address of the target hub.
`PortNum`	is the number of port whose feature is to be cleared.
`FeatureSelector`	is the USB-Specific Hub Class Feature Selector value. See Table 5 *(Hub Class Feature Selector Values)* for definition.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To clear the PORT_POWER feature on a port (Port Number 0x2) of the Hub (Hub Address 0x1), the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char HubAddr;
unsigned char PortNum;
unsigned char FeatureSelector;

HubAddr = 0x1;
PortNum = 0x2;
FeatureSelector = PORT_POWER; // Feature to be set.

svIntCmdResponse = ClearPortFeature(HubAddr, PortNum, FeatureSelector);
```

3.3.4 Host Response Functions

The host response functions report the final response of command functions to the system software through `USBPlib_ResponseGen()` function. The `USBPlib_ResponseGen()` function is called by the AT43USB370 library to return the response of a particular command function to the system software. The argument to this function specifies the kind of response for which the response is generated by the AT43USB370. Depending upon the argument, the system software calls the respective API response function which returns the response information.

Table 6 Host Response Functions

Return Type	Function Name	Arguments	Status ISR Argument
`sFinalCmdResponse`	`GetFinalResponse`	*`void`*	`H_FINAL_CMD_RESPONSE`

The following sections describe the API response functions.

3.3.4.1 GetFinalResponse ()

The AT43USB370 provides the response to the system software through the `USBPlib_ResponseGen()` function with `H_FINAL_CMD_RESPONSE` (defined in AT43USB370.h) passes as argument to the function. The system software recognizes the response by seeing this argument and calls the API response function `GetFinalResponse()` which returns the response structure.

Syntax

```
sFinalCmdResponse GetFinalResponse(void)
```

where **sFinalCmdResponse** is the structure indicating the final response as described below

```
struct sFinalCmdResponse
{
   unsigned char DevAddr;
   unsigned char EndpointAddr;
   unsigned char CmdID;
   unsigned char DT;
   unsigned int  CountXfrd;
   unsigned char Response;
   unsigned char ResponseValid;
};
```

where:

DevAddr	is the device address of the device.
EndpointAddr	is the endpoint address of the endpoint.
CmdID	is the Command ID of a particular command. This is the `CmdID` value of the `sIntCmdResponse` response structure returned while issuing the command through API Command function.
DT	is the data toggle sequence bit. It contains the toggle value sent of the last transaction associated with the command.
CountXfrd	Is the actual number of bytes transferred.
Response	indicates the kind of response returned by the transaction(s) associated with the command issued. Following reasons may be issued:

Response	Value
RESP_EP_HALTED	0x0
RESP_INTERNAL_ERR	0x3
RESP_BUF_OVERRUN	0x4
RESP_NAK_RCVD	0x6
RESP_TIME_OUT_RCVD	0x7
RESP_CMD_SUCC_EXE	0xF

The system software determines the response from the **Response** field. The system software also gets the command Id from this structure to determine as to which command this response belongs.

The Responses have the following meanings.

- *RESP_EP_HALTED*:

This Reason indicates that a serious error has occurred at the device/endpoint addressed by the command. Babble or reception of the STALL handshake can cause this from the device during a transaction.

- *RESP_INTERNAL_ERR*:

Host Controller's internal error.

- *RESP_BUF_OVERRUN*:
Packet babble has occurred. Buffer provided has run short.

- *RESP_NAK_RCVD*:
NAK is received.

- *RESP_TIME_OUT_RCVD*:
Time Out is received.

- *RESP_CMD_SUCC_EXE*:
The command has been successfully executed.

`ResponseValid`	Set to 0x1 by the AT43USB370 Library.

4 *Device Application Programming Interface*

4.1 AT43USB370 Device

AT43USB370 can operate as a standard USB device. A USB device implementation requires a generic USB functionality support for the upstream USB hosts, which includes device enumeration and data transfer. The device application runs on the system processor and makes use of this API to communicate with the AT43USB370 Device.

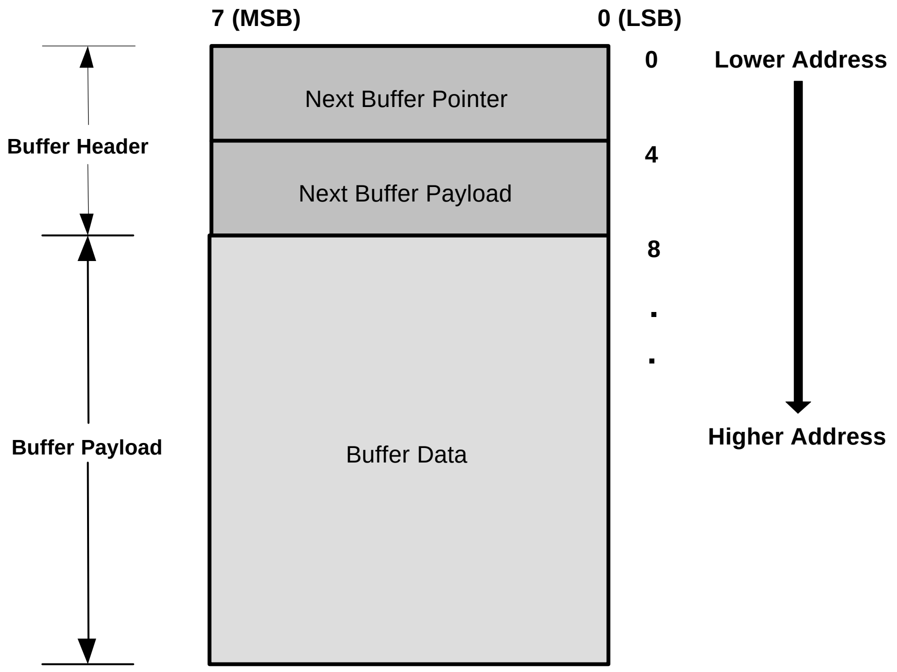
4.2 Device Buffer Structure

The data is exchanged between the device application and the AT43USB370 Library through data buffers provided by the system software. All the API functions that involve data transfer between the device application and AT43USB370's FIFOs require a pointer to the data buffer containing the data to be transmitted or received. The data buffer may be allocated from the system processor's memory while calling the API command function and freed when the final response is returned by the AT43USB370 through API response function.

The AT43USB370 allows the system software to provide a single buffer or multiple linked buffers. A single buffer may be sufficient for sending a small amount of data such as from the control endpoint of the AT43USB370 device. Multiple buffers may be required where a large contiguous memory is not available to the system software to provide data for endpoints with large Maximum Packet Sizes . In that case, various small buffers available may be linked to form a buffer linked list and provided to the AT43USB370 library. The AT43USB370 Device facilitates data transfer to/from these buffers.

For a single buffer, alignment is not required at the end of the buffer. For multiple buffers, the end of every buffer except the last one must be aligned at 4-byte boundary. At the start of each buffer there is an 8-byte header that contains the pointer and payload of the next buffer. The start of every buffer is required to be aligned at 4 byte boundary. Figure 8 shows the required format of a buffer:

Figure 8 Device Buffer Structure



"Next Buffer Payload" contains the size of the next buffer in the buffer linked list. This value excludes the 8-byte buffer header.

"Next Buffer Pointer" is a pointer to the start of the next buffer in the buffer linked list.

4.3 High Level Device API

The high level device API functions are grouped into the following three categories.

- ***Device Status Functions***
  Report various events to the system software. The events may be related to setting a particular attribute or control on the AT43USB370 device by the USB host

- ***Device Command Functions***
  Set a particular attribute of the AT43USB370 device and transfer data to/from the USB host.

- ***Device Response Functions***
  Report the responses of the command functions after they are executed by AT43USB370 device.

4.3.1 Device Header File

The AT43USB370 provides a header file (**AT43USB370.h**) to the system software through which descriptors may be loaded and various configuration constants may be set for the AT43USB370 device. These constants are described below:

Endpoint FIFO Sizes

The AT43USB370's FIFOs can be configured according to the maximum Packet Size of various endpoints by this constant.

```
#define EPx_FIFO_SIZE      (0x0080)
```

where x is endpoint number from 0 to 0xF.

Loading Descriptors

All the descriptors for the AT43USB370 Device must be specified in the following format:
The descriptors are arranged according to USB Specification in 32 bit entries as follows.

```
#define UserDescInfo  {0x01100112,0x40000000,0x0456F123,\
                       0x02010101,0x00000103,0x00370209,\
                       0xC0040101,0x00040900,0xFFFF0200,\
                       0x050705FF,0x00400281,0x05050700,\
                       0x01000001,0x01000409,0xFFFFFF02,\
                       0x81050706,0x00004002,0x01050507,\
                       0x010100}
```

String Descriptors

The string descriptors may be defined as follows:

#define String0 0x0409
#define Stringx "ATMEL Corporation Inc."

where x is the string number from 1 to 0xFF

4.3.2 Device Status Functions

These functions report various status events to the system software. When any such event is detected by the AT43USB370, API function `USBPlib_ResponseGen()` is called by the AT3USB370 library to inform the system software about the status event. The argument to this function specifies the event for which the status is generated by the AT43USB370. Depending upon the argument, the system software calls the respective status API function which returns the status information.

Table 7 Device Status Functions

Return Type	Function Name	Argument	USBPlib_ResponseGen() Argument
NA	Not Applicable	NA	D_RESET
NA	Not Applicable	NA	D_SUSPENDED
NA	Not Applicable	NA	D_RESUMED
NA	Not Applicable	NA	D_DISCONNECTED
unsigned char	DevSetConfiguration	*void*	D_SET_CONFIGURATION
sDevInterface	DevSetInterface	*void*	D_SET_INTERFACE
sDevFeature	DevSetFeature	*void*	D_SET_FEATURE
sDevFeature	DevClearFeature	*void*	D_CLEAR_FEATURE
sDevOutData	DevOutDataRcvd	*void*	D_OUT_DATA_RCVD

The following sections describe these functions/ .

4.3.2.1 D_RESET

This argument in the `USBPlib_ResponseGen()` specifies that the AT43USB370 device has been reset by USB host. There is no corresponding API status function to be called by the system software for this event.

4.3.2.2 D_SUSPENDED

This argument in the `USBPlib_ResponseGen()` specifies that the AT43USB370 device has been suspended by USB host. There is no corresponding API status function to be called by the system software for this event.

4.3.2.3 D_RESUMED

This argument in the `USBPlib_ResponseGen()` specifies that the AT43USB370 device has been resumed by USB host. There is no corresponding API status function to be called by the system software for this event.

4.3.2.4 D_DISCONNECTED

This argument in the `USBPlib_ResponseGen()` specifies that the AT43USB370 device has been disconnected from the USB host. There is no corresponding API status function to be called by the system software for this event.

4.3.2.5 DevSetConfiguration ()

This function reports that a configuration has been set on the AT43USB370 device by host. The AT43USB370 informs the system software about the event by calling `USBPlib_ResponseGen()` function with `D_SET_CONFIGURATION (`defined in `AT43USB370.h)),` passed as argument to the function`.` The system software recognizes the event by seeing this argument and calls the device API status function `DevSetConfiguration()` which returns the configuration value set by the USB host.

Syntax

`unsigned char` `DevSetConfiguration` **`(void)`**

Return Value

Returns the configuration value of the configuration set.

4.3.2.6 DevSetInterface ()

This function reports that an interface alternate setting has been set on the AT43USB370 device by USB host. The AT43USB370 informs the system software about the event by calling `USBPlib_ResponseGen()`function with `D_SET_INTERFACE (`defined in `AT43USB370.h),` passed as argument to the function`.` The system software recognizes the event by seeing this argument and calls the device API status function `DevSetInterface()` which returns the interface structure.

Syntax

```
sDevInterface DevSetInterface (void)
```

Return Value

Returns the structure **`sDevInterface`** of the following type:

```
struct sDevInterface
{
   unsigned char InterfaceNum;
   unsigned char AltSettingNum;
};
```

where:

`InterfaceNum`	is the interface number of the interface set.
`AltSettingNum`	is the alternate setting number of the interface alternate setting.

4.3.2.7 DevSetFeature ()

This function reports that a feature has been set on the AT43USB370 device by host. The AT43USB370 informs the system software about the event by calling `USBPlib_ResponseGen()` function with `D_SET_FEATURE` (defined in `AT43USB370.h`), passed as argument to the function. The system software recognizes the nature of event by seeing this argument and calls the device API status function `DevSetFeature()` which returns the feature structure containing information about the feature set by the USB Host.

Syntax

```
sDevFeature DevSetFeature (void)
```

Return Value

Returns the structure **`sDevFeature`** of the following type:

```
struct sDevFeature
{
   unsigned char Target;
   unsigned char FeatureSelector;
};
```

where:

`Target`	USB-specific value
`FeatureSelector`	USB-specific value

4.3.2.8 DevClearFeature ()

This function reports that a feature has been cleared on the AT43USB370 device by host. The AT43USB370 informs the system software about the event by calling `USBPlib_ResponseGen()` function with `D_CLEAR_FEATURE` (defined in `AT43USB370.h`), passed as argument to the function. The system software recognizes the nature of event by seeing this argument and calls the device API status function `DevClearFeature()` which returns the feature structure containing information about the feature cleared by the USB Host.

Syntax

```
sDevFeature DevClearFeature (void)
```

Return Value

Returns the structure **`sDevFeature`** of the following type:

```
struct sDevFeature
{
   unsigned char Target;
   unsigned char FeatureSelector;
};
```

where:

Target	USB-specific value
FeatureSelector	USB-specific value

4.3.2.9 DevOutDataRcvd ()

This function reports that data has been received for the AT43USB370 device not asked by the system software. The AT43USB370 informs the system software about the event by calling `USBPlib_ResponseGen()` function with `D_OUT_DATA_RCVD` (defined in `AT43USB370.h`), passed as argument to the function. The system software recognizes the event by seeing this argument and calls the device API status function `DevOutDataRcvd ()` which returns the data structure.

Syntax

```
sDevOutDataRcvd DevOutDataRcvd (void)
```

Return Value

Returns the structure **`sDevOutDataRcvd`** of the following type:

```
struct sDevOutDataRcvd
{
   unsigned char EndpointAddr;
   unsigned int  pBufStart;
   unsigned int  BufPayload;
};
```

where:

EndpointAddr	Endpoint Address of the endpoint
pBufStart	is the pointer to the start of store buffer provided by the system software for storing the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

4.3.3 Device Command Functions

This section describes the command functions provided by the AT43USB370 library to interact with the AT43USB370 device.

Table 8 Device Command Functions

Return Type	Function Name	Arguments
sIntCmdResponse	DevGetData	*unsigned char EndpointAddr,* *unsigned int *pBufStart,* *unsigned int BufPayload*
sIntCmdResponse	DevSendData	*unsigned char EndpointAddr,* *unsigned int *pBufStart,* *unsigned int BufPayload*
sIntCmdResponse	DevAbortTransfer	*unsigned char CmdId*
sIntCmdResponse	DevStallEndpoint	*unsigned char EndpointAddr*
sIntCmdResponse	DevWakeup	*void*

4.3.3.1 Return Type – Device Command Functions

All device command functions return the structure of **sIntCmdResponse** type as described below.

```
struct sIntCmdResponse
{
   unsigned char CmdID;
   unsigned char CmdStatus;
};
```

where :

CmdID	is the Command ID of this command.
CmdStatus	is the intermediate command status. One of the following values is returned:

CmdStatus	Value
CMD_IN_PROGRESS	0x0
CMD_ERROR	0x3
CMD_SUCC_EXE	0xF

CmdStatus values have the following meanings:

- *CMD_IN_PROGRESS*

Command is in progress. Final response would be delivered through the device response function GetFinalResponse() (Called by the system software in response to USBPlib_ResponseGen() function call by the AT43USB370 library with an argument D_FINAL_CMD_RESPONSE).

- *CMD_ERROR*
  Command Error has occurred. Invalid parameters specified while calling the API command function.

- *CMD_SUCC_EXE*
  Command has been successfully executed.

Note:
The system software preserves the `CmdId` to retrieve the final command response received through a device response function. The system software must only expect the final response in case it is issued **CMD_IN_PROGRESS** in the `CmdStatus` field of this structure. For all other returned values, the system software treats the response as final from the AT43USB370.

4.3.3.2 DevGetData ()

This function gets data from the host for an OUT endpoint of the AT43USB370 device.

Syntax

```
sIntCmdResponse DevGetData (unsigned char EndpointAddr,
                            unsigned int  *pBufStart,
                            unsigned int  BufPayload)
```

where:

EndpointAddr	is the endpoint address of the target endpoint.
pBufStart	is the pointer to the start of store buffer provided by the system software for storing the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

An example function call by the system software to get data for an OUT endpoint (endpoint address 0x4), would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char EndpointAddr;
unsigned int  *pBufStart;
unsigned int  BufPayload;

EndpointAddr = 0x4;
BufHeader = 0x8; // 8-byte buffer header
BufPayload = 1024; // 1K data
pBufStart = (unsigned int *) malloc (BufPayload + BufHeader);
// Allocating memory for // receiving the data.  Additional 8 bytes are
// allocated to make the buffer header according to the AT43USB370 Buffer
// Requirement.  As a single buffer is being allocated, the buffer header
// fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload

svIntCmdResponse = DevGetData (EndpointAddr, pBufStart, BufPayload);
```

4.3.3.3 DevSendData ()

This function sends data to the USB host from an IN endpoint of the AT43USB370 device.

Syntax

```
sIntCmdResponse DevSendData (unsigned char EndpointAddr,
                             unsigned int  *pBufStart,
                             unsigned int  BufPayload)
```

where:

EndpointAddr	is the endpoint address of the target endpoint.
pBufStart	is the pointer to the start of store buffer provided by the system software for storing the data.
BufPayload	is the length of the buffer provided by the system software. This value excludes the buffer header bytes.

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

An example function call by the system software to send data from an endpoint (endpoint address 0x5), would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char BufHeader;
unsigned char EndpointAddr;
unsigned int  *pBufStart;
unsigned int  BufPayload;

EndpointAddr = 0x5;
BufHeader = 0x8; // 8-byte buffer header
BufPayload = 1024; // 1K Data
pBufStart = (unsigned int *) malloc (BufPayload + BufHeader);
// Allocating memory for // receiving the data.  Additional 8 bytes are
// allocated to make the buffer header according to the AT43USB370 Buffer
// Requirement.  As a single buffer is being allocated, the buffer header
// fields are initialized to zero.

*(pBufStart) = (unsigned int ) 0x0; // next buffer pointer
*(pBufStart + 1) = (unsigned int ) 0x0; // next buffer payload
```

```
svIntCmdResponse = DevSendData (EndpointAddr, pBufStart, BufPayload);
```

4.3.3.4 DevStallEndpoint ()

This function stalls a particular endpoint of the AT43USB370 device.

Syntax

`sIntCmdResponse` `DevStallEndpoint (`**`unsigned char`** *`EndpointAddr)`*

where:

`EndpointAddr`	is the endpoint address of the endpoint to be stalled.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

An example function call by the system software to stall an endpoint (endpoint address 0x4), would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char EndpointAddr;

EndpointAddr = 0x4;
svIntCmdResponse = DevStallEndpoint(0x4);
```

4.3.3.5 DevAbortTransfer ()

This function aborts the transaction(s) associated with a particular Command ID. The function takes the command ID (`CmdId`), issued earlier by the AT43USB370 in the intermediate response structure, in return to any command function call, as an argument.

Syntax

`sIntCmdResponse` `DevAbortTransfer (`**`unsigned char`** *`CmdId`*`)`

where:

`CmdID`	is the Command ID of a particular command. This is the `CmdID` value of the `sIntCmdResponse` response structure returned while issuing the command through device API command function.

Return Value

Returns the structure of the **`sIntCmdResponse`** type.

Example

To abort the transaction with Command ID 0xA, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
unsigned char CmdID;

CmdID = 0xA; // Command ID of the command to be terminated.

svIntCmdResponse = DevAbortTransfer (CmdID);
```

4.3.3.6 DevWakeup ()

This function sends remote wakeup to the AT43USB370 device.

Syntax

sIntCmdResponse DevWakeup (**void**)

Return Value

Returns the structure of the **sIntCmdResponse** type.

Example

To wakeup the AT43USB370 device, the function call would be:

```
sIntCmdResponse svIntCmdResponse;
svIntCmdResponse = DevWakeup();
```

4.3.4 Device Response Functions

The device response functions report the final response of command functions to the system software through `USBPlib_ResponseGen()` function. The `USBPlib_ResponseGen()` function is called by the AT43USB370 library to return the response of a particular command function to the system software. The argument to this function specifies the kind of response for which the response is generated by the AT43USB370. Depending upon the argument, the system software calls the respective API response function which returns the response information.

Table 9 Device Response Functions

Return Type	Function Name	Arguments	Status ISR Argument
sDevFinalCmdResponse	DevGetFinalResponse	*void*	FINAL_CMD_RESPONSE

The following sections describe these functions.

4.3.4.1 DevGetFinalResponse ()

The AT43USB370 provides the final response to the system software through the `USBPlib_ResponseGen()` function with `D_FINAL_CMD_RESPONSE` (defined in AT43USB370.h) passed as argument to the function. The system software recognizes the response by seeing this argument and, in turn, calls the API response function `DevGetFinalResponse()` which returns the response structure.

Syntax

sDevCmdResponse DevGetFinalResponse(**void**)

Where **sDevFinalCmdResponse** is the structure indicating the final response as described below

```
struct sDevFinalCmdResponse
{
   unsigned char EndpointAddr;
   unsigned char CmdID;
   unsigned char CountXfred;
   unsigned char Response;
};
```

where:

EndpointAddr	is the endpoint address of the endpoint.
CmdID	is the Command ID of a particular command. This is the `CmdID` value of the `sIntCmdResponse` response structure returned while issuing the command through command function.
CountXfred	Is the actual number of bytes transferred
Response	indicates the kind of response returned by the transaction(s) associated with the command issued. Following reasons may be issued:

Response	**Value**
RESP_INTERNAL_ERR	0x3
RESP_CMD_SUCC_EXE	0Xf

The system software determines the response from the **Response** field. The system software also gets the command ID from this structure to determine as to which command this response belongs.

The responses have the following meanings.

- *RESP_INTERNAL_ERR*:

Device controller's internal error.

- *RESP_CMD_SUCC_EXE*:

The command has been successfully executed.

5 Low Level Host Application Programming Interface

5.1 Low Level Host API

This section describes the low level API of the AT43USB370 Host. This API consists of Host commands and Host requests.

- ***Host Commands***
  These are system software initiated activities/transactions for the AT43USB370. The commands are issued by the system software. Certain values known as command parameters need to be specified in the command registers by the system software while issuing a command to the AT43USB370.

- ***Host Requests***
  These are AT43USB370 initiated activities/transactions for the system software. The requests are issued by the AT43USB370. Certain values known as request parameters are specified in the request registers by the AT43USB370 while issuing a request to the system software.

5.1.1 Host Command Set

This section describes various commands used by the system software to communicate with the AT43USB370's host firmware. A command may be issued by the system software for data, control or status information exchange. Depending upon the command, some command parameters need to be specified in the command registers.

To issue a command to the AT43USB370, the system software activates the INTR_I interrupt line. The AT43USB370 enters the PIO scan mode and the system software can write the command data in the specified registers. When the system microprocessor ends the PIO transaction, the AT43USB370 reads and acts upon the command.

After issuing a command, the system software reads the USBP_CMDID_REG to get a unique Command ID issued by the AT43USB370 for this command. The status (completion, failure or other information such as data) for this command is then exchanged based on this Command ID. After the execution of the command, the AT43USB370 issues a request to the system microprocessor and provides the Command ID of the executed command in the USBP_EXECMDID_REG register. In this way, the system software can keep track of the completion status of various commands that it has issued to the AT43USB370. The following sections describe the host commands.

Note: The naming convention of the command names is as follows.

1- Various parts in the command name are separated by underscore '_'.
2- All commands are preceded by '**CMD'** prefix**.**
3- After the prefix, the command name contains the identification literal **'H'** which indicates the command is issued by the System software.
4- The command name is followed by the prefix and identification literal.

Host Command Codes

Every command is identified by a unique command code. These command codes are shown in the Table 10. The command codes are written in the **SYSP_CMD_REG** register by the system software while issuing commands to the AT43USB370.

Table 10 Host Command Codes

Command Name	Code
CMD_H_RESET_DEV	H'5C'
CMD_H_SUSPEND_DEV	H'5A'
CMD_H_RESUME_DEV	H'5B'
CMD_H_TX_DMA	H'51'
CMD_H_RX_DMA	H'53'
CMD_H_TX_DIRF	H'56'
CMD_H_RX_DIRF	H'57'
CMD_H_SNDBUF_ALLOC	H'30'
CMD_H_GETBUF_ALLOC	H'31'
CMD_H_ABORT_XFR	H'70'
CMD_H_SWITCH_TO_DEV	H'1A'

The following section describes the host command set.

5.1.1.1 CMD_H_RESET_DEV

This command is issued by the system software to reset a device. The table below mentions the command parameters to be specified while issuing this command

REGISTER	DESCRIPTION
SYSP_CMD_REG	CMD_H_RESET_DEV
SYSP_DEVADDR_REG	Device address of the device to be reset.

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

5.1.1.2 CMD_H_SUSPEND_DEV

This command is issued by the system software to suspend a device. The table below mentions the command parameters to be specified while issuing this command:

REGISTER	DESCRIPTION
SYSP_CMD_REG	CMD_H_SUSPEND_DEV
SYSP_DEVADDR_REG	Device address of the device to be suspended.

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

5.1.1.3 CMD_H_RESUME_DEV

This command is issued by the system software to resume a device. The table below specifies the command parameters to be specified while issuing this command:

REGISTER	**DESCRIPTION**
SYSP_CMD_REG	CMD_H_RESUME_DEV
SYSP_DEVADDR_REG	Device address of the device to be resumed.

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

5.1.1.4 CMD_H_TX_DMA

This command is issued by the system software to send data to OUT endpoints of USB device through DMA interface. The table below specifies the command parameters to be specified while issuing this command

REGISTER	**DESCRIPTION**	
SYSP_CMD_REG	CMD_H_TX_DMA	
SYSP_DEVADDR_REG	Device address of the device to which data is sent.	
SYSP_PKTSIZE_REG [7:0]	Packet Size (LSB)	This register specifies the packet size for each transaction to be sent for this command. The system software needs to specify the packet size **only** for Isochronous endpoints in this register. For Control, Bulk and Interrupt endpoints, this register **must** be set to zero while issuing the command.
SYSP_PKTSIZE_REG [15:8]	Packet Size (MSB)	
SYSP_EPATT_REG	Endpoint attributes. See bitmap below	
SYSP_SNDADDR_REG [7:0]	Send Address (LSB)	Start address of the buffer to be transmitted.
SYSP_SNDADDR_REG [15:8]	Send Address	
SYSP_SNDADDR_REG [23:16]	Send Address	
SYSP_SNDADDR_REG [31:24]	Send Address (MSB)	
SYSP_SNDCNT_REG [7:0]	Send Count (LSB)	Size of the buffer to be transmitted.
SYSP_SNDCNT_REG [15:8]	Send Count	

SYSP_SNDCNT_REG [23:16]	Send Count	
SYSP_SNDCNT_REG [31:24]	Send Count (MSB)	

The following table mentions the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap		
Bit	Field	Description
3:0	EP ADDR	Endpoint address of the target OUT endpoint
5:4	PKT TYPE	0x0 OUT/IN Packet
		0x3 SETUP Packet
31:6	RS	This field is reserved and must be reset to zero.

This command is followed by the REQ_H_START_TX_DMA and REQ_H_DMA_COMPLETE requests by the AT43USB370 to transfer data. The data is transferred from the system processor's memory to AT43USB370 memory (FIFOs).

The AT43USB370 issues the REQ_H_NEW_BUFFER request to the system software when the end of the buffer provided by the system software is reached. The system software, in response, issues CMD_H_SNDBUF_ALLOC command to provide further buffer. If the buffer count of the buffer provided is not zero, the command processing continues, otherwise the AT43USB370 assumes that the system software has no more data to send for this command and the command is terminated by the AT43USB370 when the current buffer expires.

The AT43USB370 generates one of the following requests in response to this command:

- **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

- **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

5.1.1.5 CMD_H_RX_DMA

This command is issued by the system software to get data from IN endpoint of USB device through DMA interface. The table below specifies the command parameters to be specified while issuing this command:

REGISTER	**DESCRIPTION**	
SYSP_CMD_REG	CMD_H_RX_DMA	
SYSP_DEVADDR_REG	Device address of the device from which data is received.	
SYSP_PKTSIZE_REG [7:0]	Packet Size (LSB)	This register specifies the packet size for each transaction to be sent for this command. The system software needs to specify the packet size **only** for Isochronous endpoints in this register. For Control, Bulk and Interrupt endpoints, this register **must** be set to zero while issuing the command.
SYSP_PKTSIZE_REG [15:8]	Packet Size (MSB)	

SYSP_EPATT_REG	Endpoint attributes. See bitmap below	
SYSP_GETADDR_REG [7:0]	Get Address (LSB)	Start address of the buffer to store data.
SYSP_GETADDR_REG [15:8]	Get Address	
SYSP_GETADDR_REG [23:16]	Get Address	
SYSP_GETADDR_REG [31:24]	Get Address (MSB)	
SYSP_GETCNT_REG [7:0]	Get Count (LSB)	Size of the buffer to be store data.
SYSP_GETCNT_REG [15:8]	Get Count	
SYSP_GETCNT_REG [23:16]	Get Count	
SYSP_GETCNT_REG [31:24]	Get Count (MSB)	

The following table mentions the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap			
Bit	Field	Description	
3:0	EP ADDR	Endpoint address of the target OUT endpoint	
5:4	PKT TYPE	0x0	OUT/IN Packet
		0x3	SETUP Packet
31:6	RS	This field is reserved and must be reset to zero.	

This command is followed by the REQ_H_START_RX_DMA and REQ_H_DMA_COMPLETE requests by AT43USB370 to transfer the data. The data is transferred from the AT43USB370's memory (FIFOs) to system processor's memory.

The AT43USB370 issues the REQ_H_NEW_BUF request to the system software when the end of this buffer is reached. The system software, in response issues CMD_H_GETBUF_ALLOC command to provide further buffer. If the buffer count of the buffer provided is not zero, the command processing continues, otherwise AT43USB370 assumes that system software has no more data to get for this command and the command is terminated by the AT43USB370 when current buffer expires.

The AT43USB370 generates one of the following requests in response to this command:

- **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

- **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

5.1.1.6 CMD_H_TX_DIRF

This command is issued by the system software to send data to OUT endpoints of USB device through Direct FIFO interface. The table below specifies the command parameters to be specified while issuing this command

REGISTER	**DESCRIPTION**
SYSP_CMD_REG	CMD_H_TX_DIRF

SYSP_DEVADDR_REG	Device address of the device to which data is sent.	
SYSP_PKTSIZE_REG [7:0]	Packet Size (LSB)	This register specifies the packet size for each transaction to be sent for this command. The system software needs to specify the packet size **only** for Isochronous endpoints in this register. For Control, Bulk and Interrupt endpoints, this register **must** be set to zero while issuing the command.
SYSP_PKTSIZE_REG [15:8]	Packet Size (MSB)	
SYSP_EPATT_REG	Endpoint attributes. See bitmap below	
SYSP_DIRFCNT_REG [7:0]	Direct FIFO Count (LSB)	Count of data to be transmitted.
SYSP_DIRFCNT_REG [15:8]	Direct FIFO Count (MSB)	

The following table mentions the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap		
Bit	Field	Description
3:0	EP ADDR	Endpoint address of the target OUT endpoint
5:4	PKT TYPE	0x0 OUT/IN Packet
		0x3 SETUP Packet
31:6	RS	This field is reserved and must be reset to zero.

The system software writes the data on location 0xFF until all the byte count specified in the SYSP_DIRFCNT_REG register is written in the AT43USB370's FIFO.

5.1.1.7 CMD_H_RX_DIRF

This command is issued by the system software to receive data from IN endpoints of USB device through Direct FIFO interface. The table below specifies the command parameters to be specified while issuing this command

REGISTER	**DESCRIPTION**
SYSP_CMD_REG	CMD_H_RX_DIRF
SYSP_DEVADDR_REG	Device address of the device to which data is sent.
SYSP_EPATT_REG	Endpoint attributes. See bitmap below

The following table mentions the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap		
Bit	Field	Description
3:0	EP ADDR	Endpoint address of the target OUT endpoint
5:4	PKT TYPE	0x0 OUT/IN Packet
		0x3 SETUP Packet
31:6	RS	This field is reserved and must be reset to zero.

The AT43USB370 issues a REQ_H_DIRF_RCVD request to the system software to let it read the data from FIFO.

5.1.1.8 CMD_H_SNDBUF_ALLOC

This command is issued by the system software in response to REQ_H_NEW_BUF request. The command is linked to a particular CMD_H_TX_DMA command by the AT43USB370. The table below mentions the command parameters to be specified while issuing this command:

REGISTER	DESCRIPTION	
SYSP_CMD_REG	CMD_H_SNDBUF_ALLOC	
SYSP_DEVADDR_REG	Device address of the device for which buffer is allocated.	
SYSP_SNDADDR_REG [7:0]	Send Address (LSB)	Start address of the buffer to be transmitted.
SYSP_SNDADDR_REG [15:8]	Send Address	
SYSP_SNDADDR_REG [23:16]	Send Address	
SYSP_SNDADDR_REG [31:24]	Send Address (MSB)	
SYSP_SNDCNT_REG [7:0]	Send Count (LSB)	Size of the buffer to be transmitted.
SYSP_SNDCNT_REG [15:8]	Send Count	
SYSP_SNDCNT_REG [23:16]	Send Count	
SYSP_SNDCNT_REG [31:24]	Send Count (MSB)	

Note:
REQ_H_CMD_SUCC_EXE/ REQ_H_CMD_FAILED request is **not** issued in response to this command. The AT43USB370 links the buffer provided to a particular command issued earlier. A zero length buffer provided through this command indicates to the AT43USB370 that no more buffer is linked to this command and the AT43USB370 issues the command status once the existing buffer is expired.

5.1.1.9 CMD_H_GETBUF_ALLOC

This command is issued by the system software in response to REQ_H_NEW_BUF request. The command is linked to a particular CMD_H_RX_DMA command by the AT43USB370. The table below mentions the command parameters to be specified while issuing this command:

REGISTER	DESCRIPTION	
SYSP_CMD_REG	CMD_H_GETBUF_ALLOC	
SYSP_DEVADDR_REG	Device address of the device for which buffer is allocated.	
SYSP_GETADDR_REG [7:0]	Get Address (LSB)	Start address of the buffer to store data.
SYSP_GETADDR_REG [15:8]	Get Address	
SYSP_GETADDR_REG [23:16]	Get Address	
SYSP_GETADDR_REG [31:24]	Get Address (MSB)	
SYSP_GETCNT_REG [7:0]	Get Count (LSB)	Size of the buffer to store data.
SYSP_GETCNT_REG [15:8]	Get Count	
SYSP_GETCNT_REG [23:16]	Get Count	
SYSP_GETCNT_REG [31:24]	Get Count (MSB)	

Note:
REQ_H_CMD_SUCC_EXE/ REQ_H_CMD_FAILED request is **not** issued in response to this command. The AT43USB370 links the buffer provided to a particular command issued earlier. A zero length buffer provided through this command indicates to the AT43USB370 that no more buffer is linked to this command and the AT43USB370 issues the command status once the existing store buffer is expired.

5.1.1.10 CMD_H_SWITCH_TO_DEV

This command is used by the system software to switch to the AT43USB370's Device
mode. The following table mentions the command parameters specified by the system software while issuing this command :

REGISTER	DESCRIPTION
SYSP_CMD_REG	CMD_H_SWITCH_TO_DEV

When this command is received and AT43USB370 is running in the host mode, all the host mode status is reset, any commands/requests in progress are discarded and the AT43USB370 starts operation in device mode. The command is invalid in case the AT43USB370 is already operating in the device mode and has no effect in operation in such a case.
If the mode is successfully switched, the status of this command is issued by AT43USB370 through **REQ_H_CMD_SUCC_EXE** request. The AT43USB370 acts in the device mode onwards till a further mode switch command is received from the system software.

5.1.2 Host Request Set

This section describes the host requests. **Request** means a AT43USB370 initiated transaction/activity for the system software. A request be issued by for data, control or status information exchange. Depending upon the request, some request parameters need to be specified in the request registers.

To issue a request to the system software, the AT43USB370 writes the request parameters in the specified registers and issues an interrupt to the system microprocessor (INTR_O). Depending upon the request, the system software's ISR (Interrupt Service Routine) reads the respective request registers through the PIO interface.

Note: The naming convention of the requests is as follows.

1- Various parts in the request name are separated by underscore '_'.
2- All request are preceded by '**REQ** prefix**.**
3- After the prefix, the request name contains the identification literal **'H'** which indicates the request is issued by the AT43USB370 host.
4- The request name is followed by the prefix and identification literal.

Host Request Codes

Every request is identified by a unique request code. These request codes are shown in Table 11. The request codes are written in the **USBP_REQ_REG** register by the AT43USB370 host while issuing a request to the system software.

Table 11 Host Request Codes

Request Name	Code
REQ_H_DEV_CONNECTED	H'91'
REQ_H_DEV_DISCONNECTED	H'92'
REQ_H_CMD_SUCC_EXE	H'90'
REQ_H_CMD_FAILED	H'93'
REQ_H_NEW_BUF	H'95'
REQ_H_START_TX_DMA	H'20'
REQ_H_START_RX_DMA	H'21'
REQ_H_DMA_COMPLETE	H'22'
REQ_H_DIRF_RCVD	H''58'

The following section describes these requests

5.1.2.1 REQ_H_DEV_CONNECTED

This request is given by the AT43USB370 when a device is connected to the AT43USB380 and enumerated by it. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION	
USBP_REQ_REG	REQ_H_DEV_CONNECTED	
USBP_DEVADDR_REG	Device address of the device connected.	
USBP_CLASSCD_REG	Interface Class Code.	
USBP_HUBADDR_REG	Device address of the hub to which the device is connected. If the device is connected to the root port, this field contains 0x0.	
USBP_PORTNUM_REG	Port Number of the Hub to which the device is connected. If the device is connected to the root port, this field will contain 0x0.	
USBP_RELNUM_REG [7:0]	Release Number (LSB)	**bcdUSB** value of the device descriptor.
USBP_RELNUM_REG [15:8]	Release Number (MSB)	
USBP_VENDID_REG [7:0]	Vendor ID (LSB)	**VendorID** value of the device descriptor.
USBP_VENDID_REG [15:8]	Vendor ID (MSB)	
USBP_PRODID_REG [7:0]	Product ID (LSB)	**ProdID** value of the device descriptor.
USBP_PRODID_REG [15:8]	Product ID (MSB)	

5.1.2.2 REQ_H_DEV_DISCONNECTED

This request is given by the AT43USB370 when a device is disconnected from the AT43USB380. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_H_DEV_DISCONNECTED
USBP_DEVADDR_REG	Device address of the device disconnected.

5.1.2.3 REQ_H_CMD_SUCC_EXE

This request is generated by the AT43USB370 when a command has been successfully executed. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_H_CMD_SUCC_EXE
USBP_DEVADDR_REG	Device address of the device to which the request belongs.
USBP_EXECMDID_REG	Command ID of the command executed.

5.1.2.4 REQ_H_CMD_FAILED

This request is given by the AT43USB370 if a command cannot be successfully executed by the AT43USB370.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_H_CMD_FAILED
USBP_DEVADDR_REG	Device address of the device to which the request belongs.
USBP_EXECMDID_REG	Command ID of the command executed
USBP_REQPRM0_REG	Failure Reason. Reason of command failure. See the following table for reasons

The USBP_REQPRM0_REG register contains one of the following values indicating the reason for command failure:

Failure Reason	Value	Description
STATUS_STALL	0x1	This status indicates that stall is received in response to a particular transaction for this command.
STATUS_NAK	0x2	This status indicates that NAK is received in response to a particular transaction for this command.
STATUS_TIMEOUT	0x3	This status indicates that a transaction for this command is timed out.

5.1.2.5 REQ_H_NEW_BUF

This request is given by the AT43USB370 when, during the command processing, the memory buffer provided by the system software is about to be expired. The following table mentions the request parameters specified by AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_H_NEW_BUFF
USBP_DEVADDR_REG	Device address of the device.
USBP_EXECMDID_REG	Command ID of the command for which buffer is requested

The system software determines the command for which the buffer is being requested by reading the USBP_EXECMDID_REG register and provides the buffer in its linked list to the AT43USB370 using the CMD_H_SNDBUF_ALLOC or CMD_H_GETBUF_ALLOC command. If the system software has no data to send/receive for this particular command, it should provide a zero-length buffer in response to this request . For a non-zero buffer provided by the system software, the AT43USB370 continues with the execution of the command.

5.1.2.6 REQ_H_START_TX_DMA

This request is generated by the AT43USB370 to start DMA TX transfer to transfer the data from system processor's memory to the AT43USB370's memory (FIFOs). The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**	
USBP_REQ_REG	REQ_H_START_TX_DMA	
USBP_XFRADDR_REG [7:0]	Transfer Address (LSB)	Start address for the DMA transfer.
USBP_XFRADDR_REG [15:8]	Transfer Address	
USBP_XFRADDR_REG [23:16]	Transfer Address	
USBP_XFRADDR_REG [31:24]	Transfer Address (MSB)	
USBP_XFRCNT_REG [7:0]	Transfer Count (LSB)	Size in bytes of the data to be transferred through DMA.
USBP_XFRCNT_REG [15:8]	Transfer Count (MSB)	

5.1.2.7 REQ_H_START_RX_DMA

This request is generated by the AT43USB370 to start DMA RX transfer to transfer the data from the AT43USB370's memory (FIFOs) to system processor's memory. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**	
USBP_REQ_REG	REQ_H_START_RX_DMA	
USBP_XFRADDR_REG [7:0]	Transfer Address (LSB)	Start address for the DMA transfer.
USBP_XFRADDR_REG [15:8]	Transfer Address	
USBP_XFRADDR_REG [23:16]	Transfer Address	
USBP_XFRADDR_REG [31:24]	Transfer Address (MSB)	
USBP_XFRCNT_REG [7:0]	Transfer Count (LSB)	Size in bytes of the data to be transferred through DMA.
USBP_XFRCNT_REG [15:8]	Transfer Count (MSB)	

5.1.2.8 REQ_H_DMA_COMPLETE

This request is issued by the AT43USB370 when a DMA transfer has been completed. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**
USBP_REQ_REG	REQ_H_DMA_COMPLETE

5.1.2.9 REQ_H_DIRF_RCVD

This request is generated by the AT43USB370 to let the system software read data from the AT43SUB370's FIFO. The data will be transferred from the AT43USB370's memory (FIFOs) to system processor's memory. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**	
USBP_REQ_REG	REQ_H_DIRF_RCVD	
SYSP_DIRFCNT_REG [7:0]	FIFO Count (LSB)	Count in bytes to read from FIFO.
SYSP_DIRFCNT_REG [15:8]	FIFO Count (MSB)	

The system processor reads the number of bytes specified in the SYSP_DIRFCNT_REG register from location 0xFF.

6 *Low Level Device Application Programming Interface*

6.1 Low Level Device API

This section describes the low level API of AT43USB370 device. This API consists of device commands and device requests.

- ***Device Commands***
  These are system software initiated activity/transactions for the AT43USB370. The device commands are issued by the system software. There are certain values known as command parameters which need to be specified in command registers by the system software while issuing a command to the AT43USB370.

- ***Device Requests***
  These are AT43USB370 initiated activity/transactions for the system software. The device requests are issued by the AT43USB370. There are certain values known as request parameters which need to be specified in the request registers by the AT43USB370 while issuing a request to the system software.

6.1.1 Device Command Set

The following table shows the commands that may be issued by the system software to the AT43USB370.

Note: The naming convention of the commands is as follows.

1- Various parts in the command name are separated by underscore '_'.
2- All commands are preceded by '**CMD'** prefix**.**
3- After the prefix, the command name contains the identification literal **'D'** which indicates the command is issued by the system software.
4- The command name is followed by the prefix and identification literal.

Device Command Codes

Every command is identified by a unique command code. These command codes are shown in the Table 12. The command codes are written in the **SYSP_CMD_REG** register by the system software while issuing a command to the AT43USB370.

Table 12 Device Command Codes

Command Name	**Code**
CMD_D_LOAD_DESCRIPTOR	H'30'
CMD_D_SEND_USB_IN_DATA	H'31'
CMD_D_RX_BUF_ALLOC	H'32'
CMD_D_SWITCH_TO_HOST	H'34'

The following section describes the device commands.

6.1.1.1 CMD_D_LOAD_DESCRIPTOR

This command is issued by the system software to load the descriptors from system processor's memory to the AT43USB370's memory (FIFOs). The following table mentions the command parameters specified by the system software while issuing this command:

REGISTER	**DESCRIPTION**	
SYSP_CMD_REG	CMD_H_LOAD_DESCRIPTOR	
SYSP_DEVADDR_REG	H'FF'	
SYSP_SNDADDR_REG [7:0]	Send Address (LSB)	Start address of the buffer containing descriptors.
SYSP_SNDADDR_REG [15:8]	Send Address	
SYSP_SNDADDR_REG [23:16]	Send Address	
SYSP_SNDADDR_REG [31:24]	Send Address (MSB)	
SYSP_SNDCNT_REG [7:0]	Send Count (LSB)	Size of the descriptor buffer
SYSP_SNDCNT_REG [15:8]	Send Count	
SYSP_SNDCNT_REG [23:16]	Send Count	
SYSP_SNDCNT_REG [31:24]	Send Count (MSB)	

The system software must writes all the device descriptors in a single buffer in the format specified in the following section and gives the start address and count of the descriptor buffer to the AT43USB370 in the Send Address Register and Send Count Register respectively.

The AT43USB370 issues REQ_D_START_TX_DMA and then REQ_D_DMA_COMPLETE requests to transfer the descriptors to its memory (FIFOs).

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

Load Descriptor Format

The AT43USB370 requires a specific format from the system software to provide these descriptors.

Byte	**Description**
0	Total Enabled Endpoints
1	CORE_CONFIG in the following format: Bit2: DEV_SPEED (1 Full /0 Low) Bit7: HOST/DEV (0 for HOST, 1 for DEV) All others reserved for time being.
2	Configuration Descriptor 0 Start Address Lo (To be assigned by this Specification Later)
3	Configuration Descriptor 0 Start Address Hi (To be assigned by this Specification Later)
4	Configuration Descriptor 0 Size Low (Total Length)
5	Configuration Descriptor 0 Size Hi

```
For (each Enabled Endpoint 6 byte structure)
{
        Byte 0:EpX_Addr
        Byte 1:EpX_Ctrl_Register
        Byte 2:EpX_Thold_Register
        Byte 3:EpX_Maxp_Lo
        Byte 4:EpX_Maxp_Hi
        Byte 5: Reserved
}
```

- Then List of Descriptors is to be given in following order:

```
    Device (Std.)   *
    Device (CS)   (optional - may be more than 1)
    Configuration (Std.)   *
    Configuration (CS) (optional - may be more than 1)
    For (All Standard interfaces)
    {
        if (HID Device)
        {
                Interface (Std.) (No alternate settings)
                Interface (CS)   (optional - may be more than 1)
                HID (One Descriptor per interface)
                For (all Endpoints)
                {
                        Endpoint (Std.)
                        Endpoint (CS) (optional - may be more than 1)
                }
        }
        else
        {
                Interface (Std.)     (No Alternate settings supported)
                Interface (CS)   (optional - may be more than 1)
                For (all Endpoints)
                {
                        Endpoint (Std.)
                        Endpoint (CS) (optional - may be more than 1)
                }
        }           //  end of else
    }

    if (HID Device)
    {
        For (each Interface)
        {
                Report    *
        }
    }
    String of Index 00   *         (only English Language supported)
    For (All Strings whose indices are specified in earlier descriptors)
    {
        String Descriptors * of all indices in same order in which they appeared, i.e.  possibly
```

```
                iManufacturer (if index is nonzero)
                iProduct   (if index is nonzero)
                iSerialNo  (if index is nonzero)
                For (all Configurations which give nonzero index for iConfiguration)
                {
                        iConfiguration (only standard configuration)
                        For (all interfaces which give nonzero index for iInterface)
                                iInterface (only standard interface)
                }
    }
```

NOTE:
An asterisk '*' following the name of a descriptor means that this descriptor must be written from a new WORD. In case the previous descriptor ended on the lower byte in previous WORD, the higher byte in that WORD should be left blank (e.g. write 00h there) and this descriptor must be written from the first byte of a new WORD. Contrary to this, descriptors not marked '*' must be written in continuous order in memory without leaving any blanks in between.

6.1.1.2 CMD_D_SEND_USB_IN_DATA

This command is issued by the system software to provide data for the IN data endpoint or control endpoint of the device so it can be sent to the USB host by the AT43USB370. The following table mentions the command parameters specified by the system software while issuing this command:

REGISTER	DESCRIPTION	
SYSP_CMD_REG	CMD_D_SEND_USB_IN_DATA	
SYSP_DEVADDR_REG	H'FF'	
SYSP_PKTSIZE_REG [7:0]	Packet Size (LSB)	These registers specify the maximum packet size of the target endpoint of the device.
SYSP_PKTSIZE_REG [15:8]	Packet Size (MSB)	
SYSP_CMDPRM0_REG	Status/Stall. See the description below	
SYSP_EPATT_REG	Endpoint attributes. See bitmap below	
SYSP_SNDADDR_REG [7:0]	Send Address (LSB)	Start address of the buffer to be transmitted.
SYSP_SNDADDR_REG [15:8]	Send Address	
SYSP_SNDADDR_REG [23:16]	Send Address	
SYSP_SNDADDR_REG [31:24]	Send Address (MSB)	
SYSP_SNDCNT_REG [7:0]	Send Count (LSB)	Size of the buffer to be transmitted.
SYSP_SNDCNT_REG [15:8]	Send Count	
SYSP_SNDCNT_REG [23:16]	Send Count	
SYSP_SNDCNT_REG [31:24]	Send Count (MSB)	

The following table describes the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap		
Bit	Field	Description

3:0	EP ADDR	Endpoint address of the target endpoint
31:4	RS	This field is reserved and must be reset to zero.

Following values are valid in the SYSP_CMDPRM0_REG Register for this command:

SYSP_CMDPRM0_REG Register Bitmap		
Value	Field	Description
0x1	SEND_STALL	Stall to be sent
0x2	SEND_STATUS	Status to be sent

If zero count is specified in the Send Count Register, the AT43USB370 checks the contents of SYSP_EPATT_REG register. Following cases may exist.

- If EP ADDR field indicates a non-zero value (indicating a data endpoint), the AT43USB370 reads the SYSP_CMDPRM0_REG register to see if SEND_STALL is to be sent from this data endpoint. If SYSP_CMDPRM0_REG register has a value not equal to SEND_STALL, a zero length packet is sent for this IN data endpoint when IN token is received for the endpoint.

- If EP ADDR field contains zero (indicating Control endpoint), the AT43USB370 reads the SYSP_CMDPRM0_REG register to see if SEND_STALL is to be sent from this endpoint. If SYSP_CMDPRM0_REG register has a value equal to SEND STALL, stall status will be sent for control endpoint until a setup is received. If SYSP_CPRM0_REG register has a value equal to SEND STATUS, a zero length IN packet will be sent by the AT43USB370. If not any of above two values are present in SYSP_CMDPRM0_REG register, a zero length IN packet will be sent from control endpoint but it will not be treated as IN status stage.

This command would be followed by REQ_D_START_TX_DMA and REQ_D_DMA_COMPLETE requests from the AT43USB370 until all requested data is successfully transferred to the AT43USB370's memory (FIFOs).

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

6.1.1.3 CMD_D_RX_BUF_ALLOC

This command is issued by the system software to provide a buffer to the AT43USB370 to store data for OUT data endpoint of the device. The following table mentions the command parameters specified by the system software while issuing this command:

REGISTER	**DESCRIPTION**
SYSP_CMD_REG	CMD_D_RX_BUF_ALLOC
SYSP_DEVADDR_REG	H'FF'

SYSP_GETADDR_REG [7:0]	Get Address (LSB)	Start address of the buffer to store data
SYSP_GETADDR_REG [15:8]	Get Address	
SYSP_GETADDR_REG [23:16]	Get Address	
SYSP_GETADDR_REG [31:24]	Get Address (MSB)	
SYSP_GETCNT_REG [7:0]	Get Count (LSB)	Size of the buffer to store data.
SYSP_GETCNT_REG [15:8]	Get Count	
SYSP_GETCNT_REG [23:16]	Get Count	
SYSP_GETCNT_REG [31:24]	Get Count (MSB)	

The out packet sent to the out endpoint of the device is stored in the buffer provided by the system software in this command. The system software must ensure adequate buffering for the said endpoint considering its Maximum Packet Size. The size of the buffer provided must be equal to or greater than the Maximum Packet Size of the addressed endpoint.

The AT43USB370 generates one of the following requests in response to this command:

– **REQ_H_CMD_SUCC_EXE**
  If the command is successfully executed.

– **REQ_H_CMD_FAILED**
  If the command execution fails. Failure reason is also presented in this request.

6.1.1.4 CMD_D_SWITCH_TO_HOST

This command is used by the system software to switch to the AT43USB370's host mode. The following table mentions the command parameters specified by the system software while issuing this command :

REGISTER	**DESCRIPTION**
SYSP_CMD_REG	CMD_D_SWITCH_TO_HOST

When this command is received and the AT43USB370 is operating in the device mode, all the device mode status is reset, any commands/requests in progress are discarded and the AT43USB370 starts operation in host mode. The command is invalid in case the AT43USB370 is already operating in the host mode and has no effect in operation in such a case.
If the mode is successfully switched, the status of this command is issued by the AT43USB370 through REQ_D_SUCC_EXE request. The AT43USB370 acts in the host mode onwards till a further mode switch command is received from the system software.

6.1.1.5 Device Request Set

This section describes the request set used by the AT43USB370 device to issue requests to system software. **Request** means AT43USB370's initiated transaction/activities to the system software. A request can be issued by the AT43USB370 for data, control or status information exchange. Depending upon the request, some request parameters need to be specified in the request registers by the AT43USB370.

To issue a request to the system software, the AT43USB370 writes the request parameters in the specified registers and issues an interrupt to the system microprocessor. Depending upon the request, the system software's ISR (Interrupt Service Routine) reads the respective request register through the PIO interface.

Note: The naming convention of the requests is as follows.

1- Various parts in the request name are separated by underscore '_'.
2- All request are preceded by '**REQ** prefix**.**
3- After the prefix, the request name contains the identification literal **'D'** which indicates the request is issued by the AT43USB370 Device.
4- The request name is followed by the prefix and identification literal.

Device Request Codes

Every request is identified by a unique request code. These request codes are shown in Table 13. These request codes are written in the **USBP_REQ_REG** register by the AT43USB370 while issuing a request to the system software.

Table 13 Device Request Codes

Request Name	**Code**
REQ_D_CMD_SUCC_EXE	H'40'
REQ_D_SETUP_RCVD	H'41'
REQ_D_OUT_DATA_RCVD	H'42'
REQ_D_RESET_RCVD	H'43'
REQ_D_SET_CONFIGURATION	H'44'
REQ_D_START_TX_DMA	H'45'
REQ_D_DMA_COMPLETE	H'46'
REQ_D_DEV_SUSPENDED	H'47'

The following section describes these requests.

6.1.1.6 REQ_D_CMD_SUCC_EXE

This request is issued by the AT43USB370 when a command has been successfully executed. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**
USBP_REQ_REG	REQ_D_CMD_SUCC_EXE
USBP_DEVADDR_REG	H'FF'
USBP_EXECMDID_REG	Command ID of the command executed

The USBP_EXECMDID_REG register contains the Command ID of the command that has been executed. Typically this request will be generated only for the following commands

- **CMD_D_LOAD_DESCRIPTOR**

When data equal to Load Count has been transferred.

- **CMD_D_SEND_USB_IN_DATA**
  When data equal to Load Count has been transferred or Stall or Status Stage (or a zero length packet if Load Count is zero and Stall or Status is not to be sent) has been sent.

6.1.1.7 REQ_D_DMA_COMPLETE

This request is issued by the AT43USB370 to inform the system software about the completion of a particular DMA transfer initiated earlier by it. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_D_DMA_COMPLETE
USBP_DEVADDR_REG	H'FF'

The system software can ignore this request if it can detect the end of DMA transfer by itself.

6.1.1.8 REQ_D_SETUP_RCVD

This request is issued by the AT43USB370 to inform the system software about the setup packet received by the AT43USB370 indicating a non-standard request. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION	
USBP_REQ_REG	REQ_D_SETUP_RCVD	
USBP_DEVADDR_REG	H'FF'	
USBP_REQPRM0_REG	bmRequestType	
USBP_REQPRM1_REG	bRequest	
USBP_REQPRM2_REG	wValue (LSB)	
USBP_REQPRM3_REG	wValue (MSB)	
USBP_REQPRM4_REG	wIndex (LSB)	
USBP_REQPRM5_REG	wIndex (MSB)	
USBP_REQPRM6_REG	wLength (LSB)	
USBP_REQPRM7_REG	wLength (MSB)	

Whenever a setup request is received and the 8 bytes of the setup request are not for a standard request, this request will be issued to the system software.

6.1.1.9 REQ_D_OUT_DATA_RCVD

This request is issued by the AT43USB370 when an OUT packet has been received for a data endpoint or control endpoint. The following table mentions the request parameters specified by AT43USB370 while issuing this request.

REGISTER	DESCRIPTION	
USBP_REQ_REG	REQ_D_OUT_DATA_RCVD	
USBP_DEVADDR_REG	H'FF'	
SYSP_EPATT_REG	Endpoint attribute. See bitmap below	
USBP_REQPRM0_REG	Count (LSB)	No. of bytes received
USBP_REQPRM1_REG	Count (MSB)	

The following table describes the SYSP_EPATT_REG register bitmap:

SYSP_EPATT_REG Register Bitmap		
Bit	Field	Description
3:0	EP ADDR	Endpoint address of the target endpoint
31:4	RS	This field is reserved and must be reset to zero.

When the system software finishes reading the request parameters for this request, AT43USB370 initiates REQ_D_RX_DMA request. The data will be transferred from AT43USB370's memory to the system processor's memory. The data is stored in the buffer provided for this endpoint by the system software through CMD_D_RX_BUF_ALLOC command. This count of data transferred to this buffer and the endpoint address is specified through this request.

6.1.1.10 REQ_D_RESET_RCVD

This request is issued by the AT43USB370 to inform the system software about reset issued by the USB host. The following table mentions the request parameters specified by AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_D_RESET_RCVD
USBP_DEVADDR_REG	H'FF'

This request will be issued whenever reset is received for this device to let the system processor perform any processor-specific tasks for reset.

6.1.1.11 REQ_D_SET_CONFIGURATION

This request is issued by the AT43USB370 when a particular configuration has been set by the host. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	DESCRIPTION
USBP_REQ_REG	REQ_D_SET_CONFIGURATION
USBP_DEVADDR_REG	H'FF'
USBP_REQPRM0_REG	Configuration value set by Host

This request will be generated by the AT43USB370 whenever a Set Configuration request is received from the USB host for AT43USB370 device.

6.1.1.12 REQ_D_DEV_SUSPENDED

This request is issued by the AT43USB370 to inform the system software about suspend issued by the host. The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**
USBP_REQ_REG	REQ_D_DEV_SUSPENDED
USBP_DEVADDR_REG	H'FF'

The AT43USB370 remains suspended unless

- There is a resume signaling from Host *Or*
- System software issues the CMD_D_SEND_USB_IN_DATA command for any endpoint of the device. In this case, the AT43USB370 device initiates remote wakeup on the bus (If Its remote wakeup feature has been set by Host). There are valid time conditions after suspend for remote wakeup to be initiated. Once the USB bus is activated through SOFs, the CMD_D_SEND_USB_IN_DATA command would be treated as it is treated normally.

6.1.1.13 REQ_D_START_TX_DMA

This request is generated by the AT43USB370 to start DMA TX transfer to transfer the data from system processor's memory to the AT43USB370's memory (FIFOs). The following table mentions the request parameters specified by the AT43USB370 while issuing this request.

REGISTER	**DESCRIPTION**	
USBP_REQ_REG	REQ_D_START_TX_DMA	
USBP_XFRADDR_REG [7:0]	Transfer Address (LSB)	Start address for the DMA transfer.
USBP_XFRADDR_REG [15:8]	Transfer Address	
USBP_XFRADDR_REG [23:16]	Transfer Address	
USBP_XFRADDR_REG [31:24]	Transfer Address (MSB)	
USBP_XFRCNT_REG [7:0]	Transfer Count (LSB)	Size in bytes of the data to be transferred through DMA.
USBP_XFRCNT_REG [15:8]	Transfer Count (MSB)	

7 *System Processor Interface Register Set*

The System Processor Interface register set is used by the AT43USB370 to interact with the system processor. The same register set is used in both the host and the function modes except where explicitly stated. All registers are 32-bit wide and require access on 4-bytes boundaries.

Reading a register for which the external system processor does not have read access will yield a zero value result. Writing to a register for which the external system processor does not have write access has no effect. For detailed usage of the registers, please refer to the AT43USB370 Software Development Guide.

Naming Convention:

The following naming convention applies to the System Processor Interface Register Set.

- Three different fields in the register name are separated by underscores '_'
- The first field in the register name is a prefix indicating the Write access identification literal:
  - USBP indicates the register is always written by the AT43USB370 USB Processor.
  - SYSP indicates the register is always written by the system processor.
- The second field in the register name indicates the functionality of the register.
- The third field in the register name is a suffix 'REG' common to all the registers.

Table 14: System Processor Interface Register Set

Address	Name	Function
0x20	USBP_REQ_REG	Request Register
0x21	SYSP_CMD_REG	Command Register
0x22	SYSP_DEVADDR_REG	Device Address Register
0x23	USBP_DEVADDR_REG	Device Address Register
0x24	USBP_CLASSCD_REG	Class Code Register
0x25	Reserved	Unused
0x26	USBP_CMDID_REG	Command ID Register
0x27	USBP_EXECMDID_REG	Executed Command ID Register
0x28	SYSP_EPATT_REG	Endpoint Attributes Register
0x29	SYSP_PKTSIZE_REG [7:0]	Packet Size Register (LSB)
0x2A	SYSP_PKTSIZE_REG [15:8]	Packet Size Register (MSB)
0x2B - x2F	Reserved	Unused
0x30	SYSP_CMDPRM0_REG	Command Parameter 0 Register
0x31	SYSP_CMDPRM1_REG	Command Parameter 1 Register
0x32	SYSP_CMDPRM2_REG	Command Parameter 2 Register
0x33	SYSP_CMDPRM3_REG	Command Parameter 3 Register
0x34	SYSP_CMDPRM4_REG	Command Parameter 4 Register
0x35	SYSP_CMDPRM5_REG	Command Parameter 5 Register
0x36	SYSP_CMDPRM6_REG	Command Parameter 6 Register
0x37	SYSP_CMDPRM7_REG	Command Parameter 7 Register

0x38	USBP_VENDID_REG [7:0]	Vendor ID Register (LSB)
0x39	USBP_VENDID_REG [15:8]	Vendor ID Register (MSB)
0x3A	USBP_PRODID_REG [7:0]	Product ID Register (LSB)
0x3B	USBP_PRODID_REG [15:8]	Product ID Register (MSB)
0x3C	USBP_RELNUM_REG [7:0]	BCD Release Number Register (LSB)
0x3D	USBP_RELNUM_REG [15:8]	BCD Release Number Register (MSB)
0x3E	USBP_HUBADDR_REG	Hub Address Register
0x3F	USBP_PORTNUM_REG	Port Number Register
0x40 – 0x47	Reserved	Unused
0x48	USBP_REQPRM0_REG	Request Parameter 0 Register
0x49	USBP_REQPRM1_REG	Request Parameter 1 Register
0x4A	USBP_REQPRM2_REG	Request Parameter 2 Register
0x4B	USBP_REQPRM3_REG	Request Parameter 3 Register
0x4C	USBP_REQPRM4_REG	Request Parameter 4 Register
0x4D	USBP_REQPRM5_REG	Request Parameter 5 Register
0x4E	USBP_REQPRM6_REG	Request Parameter 6 Register
0x4F	USBP_REQPRM7_REG	Request Parameter 7 Register
0x50	SYSP_SNDADDR_REG [7:0]	Send Data Address Register (LSB)
0x51	SYSP_SNDADDR_REG [15:8]	Send Data Address Register
0x52	SYSP_SNDADDR_REG [23:16]	Send Data Address Register
0x53	SYSP_SNDADDR_REG [31:24]	Send Data Address Register (MSB)
0x54	SYSP_SNDCNT_REG [7:0]	Send Data Count Register (LSB)
0x55	SYSP_SNDCNT_REG [15:8]	Send Data Count Register
0x56	SYSP_SNDCNT_REG [23:16]	Send Data Count Register
0x57	SYSP_SNDCNT_REG [31:24]	Send Data Count Register (MSB)
0x58 – 0x5F	Reserved	Unused
0x60	SYSP_GETADDR_REG [7:0]	Get Data Address Register (LSB)
0x61	SYSP_GETADDR_REG [15:8]	Get Data Address Register
0x62	SYSP_GETADDR_REG [23:16]	Get Data Address Register
0x63	SYSP_GETADDR_REG [31:24]	Get Data Address Register (MSB)
0x64	SYSP_GETCNT_REG [7:0]	Get Data Count Register (LSB)
0x65	SYSP_GETCNT_REG [15:8]	Get Data Count Register
0x66	SYSP_GETCNT_REG [23:16]	Get Data Count Register
0x67	SYSP_GETCNT_REG [31:24]	Get Data Count Register (MSB)
0x68 – 0x71	Reserved	Unused
0x72	SYSP_DIRFCNT_REG [7:0]	Direct FIFO Count Register (LSB)
0x73	SYSP_DIRFCNT_REG [15:8]	Direct FIFO Count Register (MSB)
0x74 –	Reserved	Unused

0x77		
0x78	USBP_XFRADDR_REG [7:0]	Transfer Address Register (LSB)
0x79	USBP_XFRADDR_REG [15:8]	Transfer Address Register
0x7A	USBP_XFRADDR_REG [23:16]	Transfer Address Register
0x7B	USBP_XFRADDR_REG [31:24]	Transfer Address Register (MSB)
0x7C	USBP_XFRCNT0_REG [7:0]	Transfer Count Register (LSB) ; DMA Mode
0x7D	USBP_XFRCNT1_REG [15:8]	Transfer Count Register (MSB)
0x7E-0xFE	Reserved	Unused
0xFF	SYSP_FIFODATA_REG	FIFO Access Register

Request Register – USBP_REQ_REG

Bit	31 … 8	7 … 0	
0x20	RS	REQ CODE	**USBP_REQ_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – REQ CODE**
  Request code of the request issued by the AT43USB370.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the request codes while issuing requests to the system processor. After power-up or reset, this register will contain the value of 0x00.

Command Register – SYSP_CMD_REG

The system processor writes in this register.

Bit	31 … 8	7 … 0	
0x21	RS	CMD CODE	**SYSP_CMD_REG**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – CMD CODE**
  Command code of the command issued by the system processor.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the system processor to write the commands codes while issuing commands to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Device Address Register – SYSP_DEVADDR_REG

Bit	31 8	7 0	
0x22	RS	DEV ADDR	**SYSP_DEVADDR_REG**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – DEV ADDR**
  Device address of the target device.

- **Bit 31:8 - RS**
  Reserved. Must be reset to zero.

This register is used by the system processor to write the address of the device for which a command is being issued to the AT43USB370. This register is only used by the AT43USB30 Host. After power-up or reset, this register will contain the value of 0x00.

Device Address Register – USBP_DEVADDR_REG

Bit	31 8	7 0	
0x23	RS	DEV ADDR	**USBP_DEVADDR_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – DEV ADDR**
  Device address of the target device.

- **Bit 31:8 - RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the address of the device for which a request is being issued to the system processor. This register is used only by the AT43USB30 Host. After power-up or reset, this register will contain the value of 0x00.

Class Code Register – USBP_CLASSCD_REG

Bit	31 8	7 0	
0x24	RS	CLASS CODE	**USBP_CLASSCD_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – CLASS CODE**
  Class code value of the device.

- **Bit 31:8 - RS**

Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the class code value while issuing a request to the system processor. This register is used only by the AT43USB30 Host. After power-up or reset, this register will contain the value of 0x00.

Command ID Register – USBP_CMDID_REG

Bit	31 … 8	7 … 0	
0x26	RS	CMD ID	**USBP_CMDID_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – CMD ID**
  Command ID of the command.
- **Bit 31:8 - RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the Command ID in response to a command issued by the system processor. After power-up or reset, this register will contain the value of 0x00.

Executed Command ID Register – USBP_EXECMDID_REG

Bit	31 … 8	7 … 0	
0x27	RS	EXE CMD ID	**USBP_EXECMDID_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – EXE CMD ID**
  Command ID of the command executed.

- **Bit 31:8 - RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the Command ID of a particular command executed by the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Endpoint Attribute Register – SYSP_EPATT_REG

Bit	31 … 6	5 … 4	3 … 0	
0x28	RS	PKT TYP E	EP ADD R	**SYSP_EPATT_REG**
Read/Write	AT43USB370 R	System Processor W		

Initial Value 0x0

- **Bit 3:0 – EP ADDR**
  Endpoint address of the target endpoint.

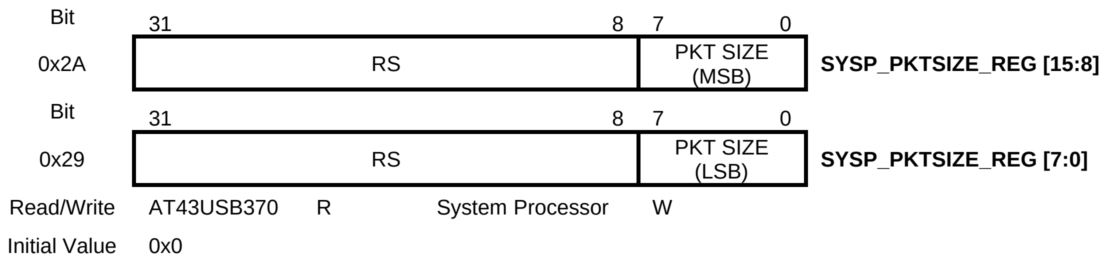
- **Bit 5:4 – PKT TYPE**
  Packet type of the packet.

00	IN Packet , OUT Packet
11	SETUP Packet

- **Bit 31:6 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify the endpoint attributes. After power-up or reset, this register will contain the value of 0x00.

Packet Size Register –SYSP_PKTSIZE_REG

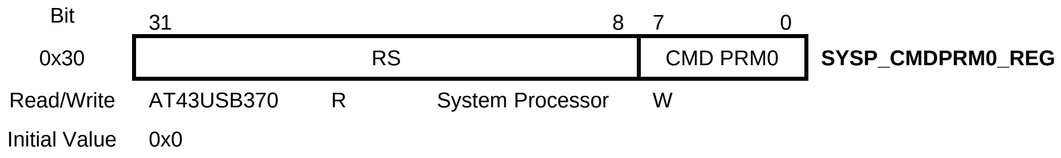


- **Bit 15:0 PCK SIZE**
  Packet size in bytes.

- **Bit 31:16 – RS**
  Reserved. Must be reset to zero by the system processor.

These registers are used by the system processor to specify the Packet Size while issuing a command to the AT43USB370. This packet size is used by the AT43USB370 Host for every transaction associated with this command. This register is used only by the AT43USB30 Host. After power-up or reset, these registers will contain the value of 0x00.

Command Parameter 0 Register – SYSP_CMDPRM0_REG



- **Bit 7:0 – CMD PRM0**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 1 Register – SYSP_CMDPRM1_REG

Bit	31 … 8		7 … 0		
0x31	RS		CMD PRM1		**SYSP_CMDPRM1_REG**
Read/Write	AT43USB370	R	System Processor	W	
Initial Value	0x0				

- **Bit 7:0 – CMD PRM1**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 2 Register – SYSP_CMDPRM2_REG

Bit	31 … 8		7 … 0		
0x32	RS		CMD PRM2		**SYSP_CMDPRM2_REG**
Read/Write	AT43USB370	R	System Processor	W	
Initial Value	0x0				

- **Bit 7:0 – CMD PRM2**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 3 Register – SYSP_CMDPRM3_REG

Bit	31 8	7 0	
0x33	RS	CMD PRM3	**SYSP_CMDPRM3_REG**
Read/Write	AT43USB370 R System Processor	W	
Initial Value	0x0		

- **Bit 7:0 – CMD PRM3**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 4 Register – SYSP_CMDPRM4_REG

Bit	31 8	7 0	
0x34	RS	CMD PRM4	**SYSP_CMDPRM4_REG**
Read/Write	AT43USB370 R System Processor	W	
Initial Value	0x0		

- **Bit 7:0 – CMD PARAMETER 4**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 5 Register – SYSP_CPRM5_REG

Bit	31 8	7 0	
0x35	RS	CMD PRM5	**SYSP_CMDPRM5_REG**
Read/Write	AT43USB370 R System Processor	W	
Initial Value	0x0		

- **Bit 7:0 – CMD PRM5**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 6 Register – SYSP_CPRM6_REG

Bit	31 8	7 0	
0x36	RS	CMD PRM6	**SYSP_CMDPRM6_REG**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – CMD PRM6**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Command Parameter 7 Register – SYSP_CPRM7_REG

Bit	31 8	7 0	
0x37	RS	CMD PRM 7	**SYSP_CMDPRM7_REG**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – CMD PRM7**
  Command-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Must be reset to zero by the system processor.

This register is used by the system processor to specify a command-specific parameter while issuing a command to the AT43USB370. After power-up or reset, this register will contain the value of 0x00.

Vendor ID Register – USBP_VENDID_REG

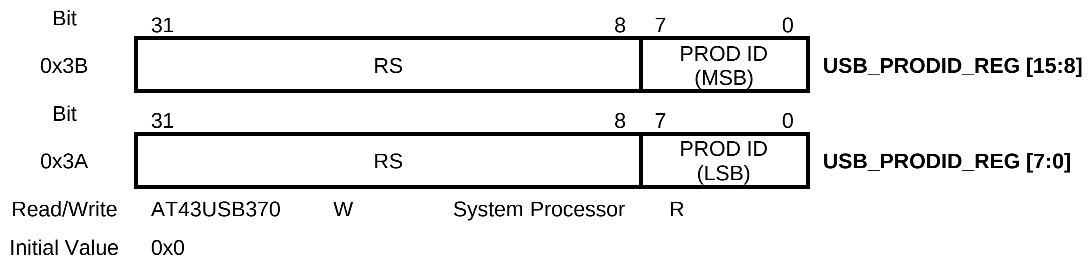
Bit	31 8	7 0	
0x39	RS	VEND ID (MSB)	**USBP_VENDID_REG [15:8]**
Bit	31 8	7 0	
0x38	RS	VEND ID (LSB)	**USBP_VENDID_REG [7:0]**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 15:0 VEND ID**
  Vendor ID of the USB device.

- **Bit 31:16 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the AT43USB370 to specify Vendor ID while issuing a request to the system software. They are only used in the host mode. After power-up or reset, They contain the value of 0x00.

Product ID Register – USBP_PRODID_REG

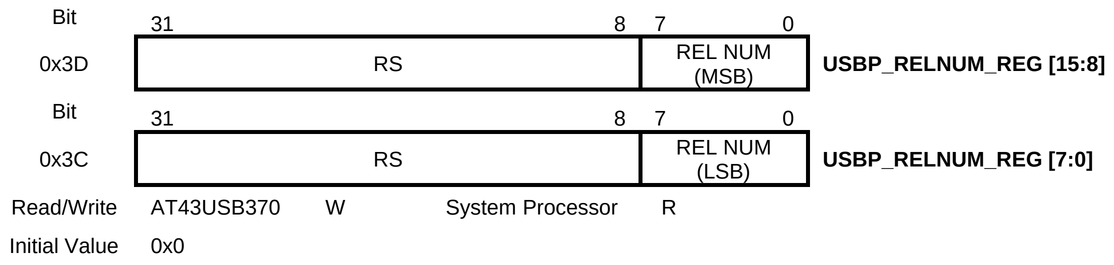


- **Bit 15:0 PROD ID**
  Product ID of the USB device.

- **Bit 31:16 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the AT43USB370 to specify the Product ID while issuing a request to the system processor. They are only used in the host mode. After power-up or reset, they will contain the value of 0x00.

Release Number Register – USBP_RELNUM_REG



- **Bit 15:0 REL NUM**
  BCD Release Number of the USB device.

- **Bit 31:16 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the AT43USB370 to specify the BCD Release Number while issuing a request to the system processor. They are used only in the host mode. After power-up or reset, they will contain the value of 0x00.

Hub Address Register – USBP_HUBADDR_REG

Bit	31	8	7	0	
0x3E	RS		HUB ADDR		**USBP_HUBADDR_REG**
Read/Write	AT43USB370 W	System Processor R			
Initial Value	0x0				

- **Bit 7:0 – HUB ADDR**
  Device address of the Hub to which the USB device is connected.
- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the device address of the hub to which a USB device is connected while issuing a request to the system processor. This register is used only by the AT43USB30 Host. After power-up or reset, this register will contain the value of 0x00.

Port Number Register – USBP_PORTNUM_REG

Bit	31	8	7	0	
0x3F	RS		PORT NUM		**USBP_PORTNUM_REG**
Read/Write	AT43USB370 W	System Processor R			
Initial Value	0x0				

- **Bit 7:0 – PORT NUM**
  Port number of the Hub to which the USB device is connected.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to write the port number of the hub to which a USB device is connected while issuing a request to the system processor. This register is used only in the host mode. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 0 Register – USBP_REQPRM0_REG

Bit	31	8	7	0	
0x48	RS		REQ PRM0		**USBP_REQPRM0_REG**
Read/Write	AT43USB370 W	System Processor R			
Initial Value	0x0				

- **Bit 7:0 – REQ PRM0**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 1 Register – USBP_REQPRM1_REG

Bit	31	8	7	0	
030x49	RS		REQ PRM1		**USBP_REQPRM1_REG**
Read/Write	AT43USB370 W	System Processor	R		
Initial Value	0x0				

- **Bit 7:0 – REQ PRM1**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 2 Register – USBP_REQPRM2_REG

Bit	31	8	7	0	
0x4A	RS		REQ PRM2		**USBP_REQPRM2_REG**
Read/Write	AT43USB370 W	System Processor	R		
Initial Value	0x0				

- **Bit 7:0 – REQ PRM2**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 3 Register – USBP_REQPRM3_REG

Bit	31	8	7	0	
0x4B	RS		REQ PRM3		**USBP_REQPRM3_REG**
Read/Write	AT43USB370 W	System Processor	R		
Initial Value	0x0				

- **Bit 7:0 – REQ PARAMETER 3**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 4 Register – USBP_REQPRM4_REG

Bit	31 8	7 0	
0x4C	RS	REQ PRM4	**USBP_REQPRM4_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – REQ PRM 4**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 5 Register – USBP_REQPRM5_REG

Bit	31 8	7 0	
0x4D	RS	REQ PRM5	**USBP_REQPRM5_REG**
Read/Write	AT43USB370 W	System Processor R	
Initial Value	0x0		

- **Bit 7:0 – REQ PARAMETER 5**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 6 Register – USBP_REQPRM6_REG

Bit	31		8	7	0	
0x4E	RS			REQ PRM6		**USBP_REQPRM6_REG**
Read/Write	AT43USB370	W	System Processor	R		
Initial Value	0x0					

- **Bit 7:0 – REQ PRM6**
  Request-specific parameter.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Request Parameter 7 Register – USBP_REQPRM7_REG

Bit	31		8	7	0	
0x4F	RS			REQ PRM7		**USBP_REQPRM7_REG**
Read/Write	AT43USB370	W	System Processor	R		
Initial Value	0x0					

- **Bit 7:0 – REQ PARAMETER 7**
  Request-specific parameter.
- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

This register is used by the AT43USB370 to specify a request-specific parameter while issuing a request to the system processor. After power-up or reset, this register will contain the value of 0x00.

Send Data Address Register – SYSP_SNDADDR_REG

Bit	31 … 8	7 … 0	
0x53	RS	SND ADDR (MSB)	**SYSP_SNDADDR_REG [31:24]**
Bit	31 … 8	7 … 0	
0x52	RS	SND ADDR	**SYSP_SNDADDR_REG [23:16]**
Bit	31 … 8	7 … 0	
0x51	RS	SND ADDR	**SYSP_SNDADDR_REG [15:8]**
Bit	31 … 8	7 … 0	
0x50	RS	SND ADDR (LSB)	**SYSP_SNDADDR_REG [7:0]**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – SND ADDR**
  Address of the buffer for sending data. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit SND ADDR.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the system processor to specify the start address of the data buffer while issuing a command to the AT43USB370 to transfer data from the system processor's memory to a USB device. After power-up or reset, this register will contain the value of 0x00.

Send Data Count Register – SYSP_SNDCNT_ REG

Bit	31 … 8	7 … 0	
0x57	RS	SND CNT (MSB)	**SYSP_SNDCNT_REG [31:24]**
Bit	31 … 8	7 … 0	
0x56	RS	SND CNT	**SYSP_SNDCNT_REG [23:16]**
Bit	31 … 8	7 … 0	
0x55	RS	SND CNT	**SYSP_SNDCNT_REG [15:8]**
Bit	31 … 8	7 … 0	
0x54	RS	SND CNT (LSB)	**SYSP_SNDCNT_REG [7:0]**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – SND CNT**
  Count of the buffer for sending data. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit SND CNT.

- **Bit 31:8 – RS**

Reserved. Reset to zero by the AT43USB370.

These registers are used by the system processor to specify the size of the data buffer while issuing a command to the AT43USB370 to transfer data from the system processor's memory to a USB device. This is the size of the buffer whose address is specified in Send Data Address Register. After power-up or reset, this register will contain the value of 0x00.

Get Data Address Register – SYSP_GETADDR_REG

Bit	31 … 8	7 … 0	
0x63	RS	GET ADDR (MSB)	**SYSP_GETADDR_REG [31:24]**
Bit	31 … 8	7 … 0	
0x62	RS	GET ADDR	**SYSP_GETADDR_REG [23:16]**
Bit	31 … 8	7 … 0	
0x61	RS	GET ADDR	**SYSP_GETADDR_REG [15:8]**
Bit	31 … 8	7 … 0	
0x60	RS	GET ADDR (LSB)	**SYSP_GETADDR_REG [7:0]**
Read/Write	AT43USB370 R	System Processor W	
Initial Value	0x0		

- **Bit 7:0 – GET ADDR**
  Address of the buffer for storing data. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit GET ADDR.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the system processor to specify the start address of the data buffer while issuing a command to the AT43USB370 to transfer data from the USB device to the system processor's memory. After power-up or reset, this register will contain the value of 0x00.

Get Data Count Register – SYSP_GETCNT_REG

Bit	31 … 8	7 … 0	
0x67	RS	GET CNT (MSB)	**SYSP_GETCNT_REG [31:24]**
0x66	RS	GETCNT	**SYSP_GETCNT_REG [23:16]**
0x65	RS	GET CNT	**SYSP_SNDCNT_REG [15:8]**
0x64	RS	GET CNT (LSB)	**SYSP_SNDCNT_REG [7:0]**

Read/Write	AT43USB370	R	System Processor	W
Initial Value	0x0			

- **Bit 7:0 – GET CNT**
  Count of the data buffer for receiving data. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit GET CNT.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the system processor to specify the size of the data buffer while issuing a command to the AT43USB370 to transfer data from the USB device to the system processor's memory. This is the size of the buffer specified in Get Data Address register. After power-up or reset, this register will contain the value of 0x00.

Direct FIFO Count Register – SYSP_DIRFCNT_REG

Bit	31 … 8	7 … 0	
0x73	RS	FIFO CNT (MSB)	**SYSP_DIRFCNT_REG [15:8]**
0x72	RS	FIFO CNT (LSB)	**SYSP_DIRFCNT_REG [7:0]**

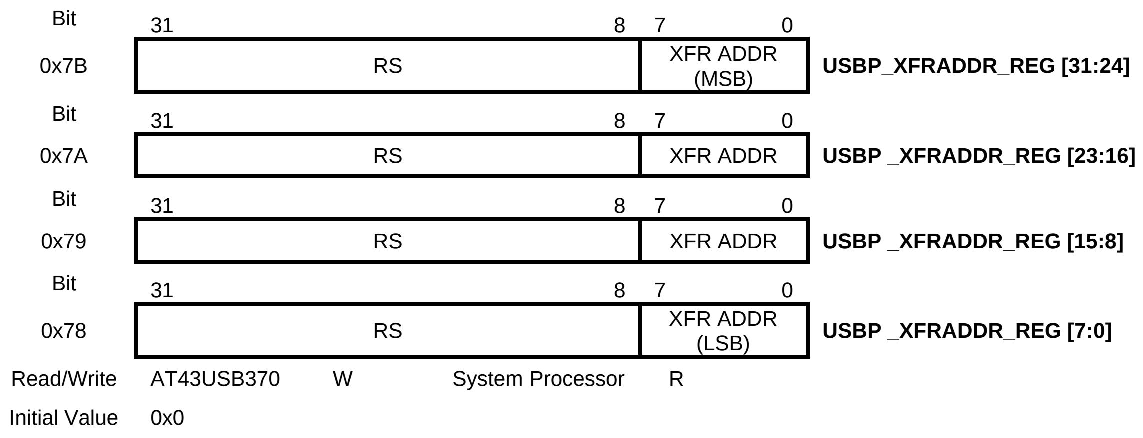
Read/Write	AT43USB370	R	System Processor	W
Initial Value	0x0			

- **Bit 7:0 – FIFO CNT**
  Count of the data to be transferred through Direct FIFO. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit FIFO CNT. Bit 31:16 of the SYSP_DIRCNT_REG default to zero.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the system processor to specify the count while issuing a command to the AT43USB370 to transfer data between the system processor's memory and the AT43USB370's FIFO. After power-up or reset, this register will contain the value of 0x00.

Transfer Address Register – USBP_XFRADDR_REG

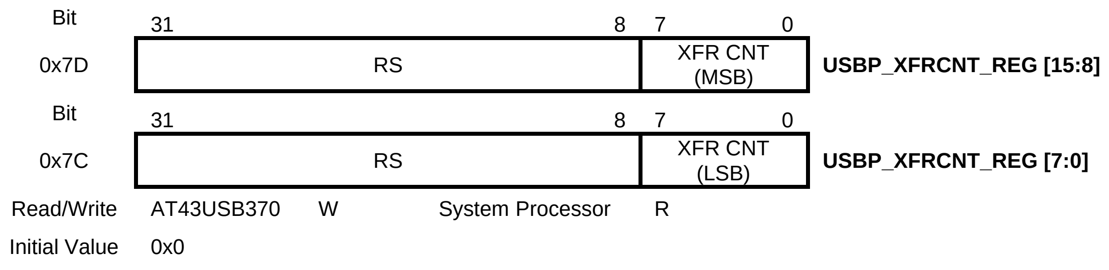


- **Bit 7:0 – XFR ADDR**
  Address for the data transfer. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit XFR ADDR.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the AT43USB370 to specify the start address of the memory while issuing a request to system processor to transfer data. After power-up or reset, this register will contain the value of 0x00.

Transfer Count Register – USBP_XFRCNT_REG



- **Bit 7:0 – XFR CNT**
  Transfer count in bytes. Bit 7:0 of each of the register locations are concatenated together to form the 32 bit XFR CNT. Bit 31:16 of the XFR CNT register default to zero.

- **Bit 31:8 – RS**
  Reserved. Reset to zero by the AT43USB370.

These registers are used by the AT43USB370 to specify the number of bytes while issuing a request to the system processor to transfer data. This register specifies the count to be transferred from the location specified in the Transfer Address register. After power-up or reset, this register will contain the value of 0x00.

FIFO Data Access Register – SYSP_FIFODATA _REG

Bit	31		8	7	0	
0x4E	FIFO Data					**SYSP_FIFODATA_REG**
Read/Write	AT43USB370	-	System Processor	R/W		
Initial Value	0x0					

- **Bit 31:0 – FIFO Data Access Register**
  Actual data to and from the FIFO.

This register is used by the system processor to either fetch the data from the AT43USB370 or push the data into the AT43USB370 FIFO. After power-up or reset, this register will contain the value of 0x00.