AN1325
APPLICATION NOTE

USING THE ST7 USB LOW-SPEED LIBRARY V4.2x

by Microcontroller Division Applications

This application note describes how to use the ST7 USB Low-Speed Library V4.2x. Starting from version 4.20 the Library supports the DFU class layer.

Caution: The DFU class and the HID class layers are not included in the Library. The Library contains only the standard USB request layers. You will find the DFU and HID layers in the ST7 USB Low-Speed DFU EvalKit firmware.

Note: In this document the names “Hiware” and “MetroWerks” refer to the same compiler manufacturer.

1 OVERVIEW

The ST7 USB Low-Speed Firmware Library is written in C language and is compatible with both Cosmic and Metrowerks compilers.

This Library provides a complete USB protocol layer for the ST7 USB Low-Speed microcontrollers (such as the ST7261, ST7262, ST7263 and ST7263B). The source code is available free to STMicroelectronics customers.

This Library is supplied in a ZIP file. Starting from version 4.10, the Library contains only the files that are common for all the projects. This new architecture has been choosen in order to better separate what is really the Library and what is related to a project. This architecture will also facilitate the upgrade of different projects with new Library versions in the future.

2 LIBRARY

2.1 DIRECTORIES ARCHITECTURE

The files which compose the Library are placed into 4 distinct directories : **Docs**, **Macro**, **Micro**, and **Usb** :

```
ST7USBLS-Library-V4.2x/
    |--- Docs/
    |--- Macro/
    |--- Micro/
    |--- Usb/
```

The files inside these directories cannot be used alone. You have to use them within a project. To do so, you have to include in your project setting files the path of these directories (Mak file for Cosmic and Default.env for Metrowerks). This way you will be able to use easily the same Library for different projects.

Example in a MAK file for Cosmic :

```
MACRO_PATH = D:\ST7USBLS-Library-V4.2x\Macro
MICRO_PATH = D:\ST7USBLS-Library-V4.2x\Micro
USB_PATH = D:\ST7USBLS-Library-V4.2x\Usb
```

Example in Default.env file for Metrowerks :

```
GENPATH=\
D:\ST7USBLS-Library-V4.2x\Macro;\
D:\ST7USBLS-Library-V4.2x\Micro;\
D:\ST7USBLS-Library-V4.2x\Usb;
```

Normally there is no need to change any of the files present in these directories. All the changes must be done inside your project files.

The following chapters will explain in detail the content of all this directories.

2.2 DOCS DIRECTORY

This directory contains all documentations related to the Library: ReleaseNote.txt, FlowChart, etc...

2.3 MACRO DIRECTORY

This directory contains all the files containing the macro definitions shared between the Library and the different projects. It contains also the Unicode table used for the definition of the String descriptors. For example you will find the description of “EnableInterrupts” and “DisableInterrupts” macro. All these files are common for Cosmic and Metrowerks compilers with the exception of the “Hidef.h” file wich is used only for Metrowerks.

2.4 MICRO DIRECTORY

This directory contains all the files which are related to the microcontrollers. You will find here all the MAP files for the ST726X microcontrollers family (ST7261, ST7262, ST7263 and ST7263B) . All these MAP files are common for Cosmic and Metrowerks compilers.

Important Note : For Metrowerks, all the H/W registers addresses are written inside the PRM files of each project. For Cosmic, all these informations are written directly inside the MAP files.

2.5 USB DIRECTORY

This directory contains the USB Library core files for Cosmic and Metrowerks compilers. These file don’t need to be changed by the customer.

This USB Library can be seen as a ToolBox where the Application picks up the “tools” it needs (calls the functions it needs). Note that there is no callback functions from the Library (the user doesn’t need to create specific functions that could be called by the Library).

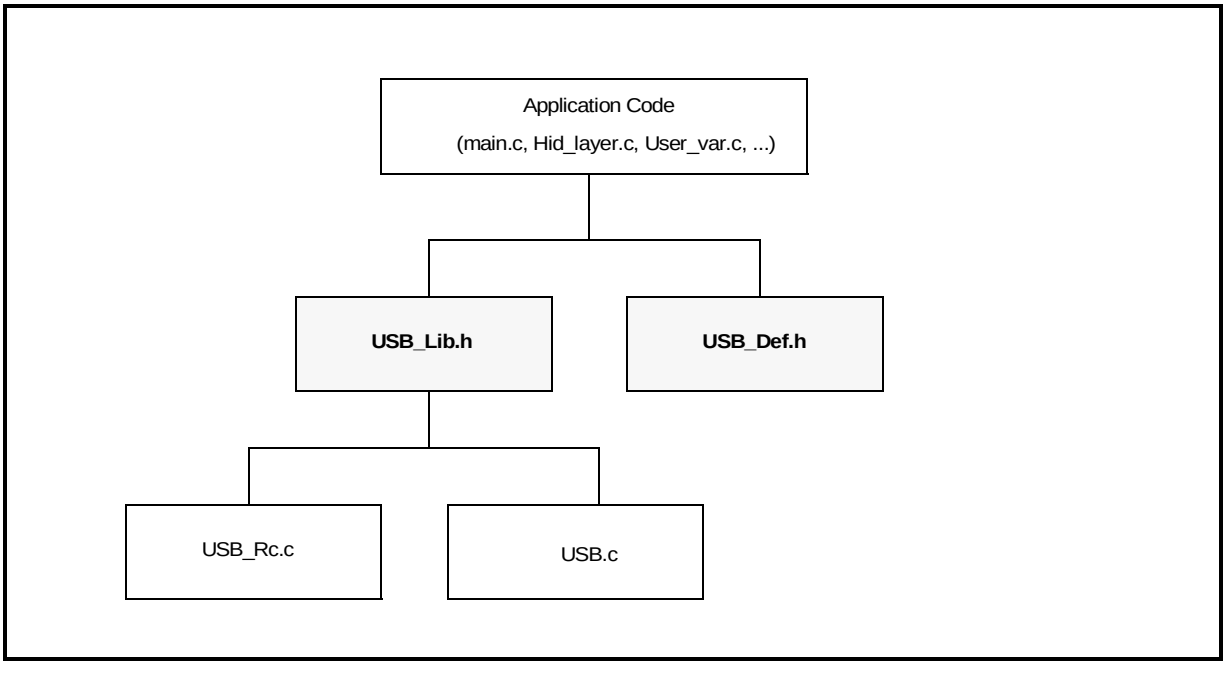
2.6 USB CORE FILES

All the following files are placed inside Usb directory. These files are really what we call THE Library.

File	Description
USB.c	Library core functions
USB.h	Library core prototypes
USB_Def.h	Description of all aliases and variables shared between the Library and the Application layer
USB_Rwu.c	Remote Wake-Up functions
USB_Rwu.h	Prototypes for Remote Wake-Up functions
USB_Lib.c	Library functions to be called by Application layer code
USB_Lib.h	Prototypes of all the Library functions to be called by the Application layer code
USB_Rc.c	Hardware abstraction layer
USB_Rc.h	Prototypes for Hardware abstraction layer
USB_Var.c	Library variables declaration
USB_Var.h	Prototypes for Library variables
USB_JumpTable_Cosmic.asm	Contains Jump to USB functions (for DFU only)
USB_JumpTable_Hiware.asm	Contains Jump to USB functions (for DFU only)

The 2 assembly files (USB_JumpTable_xxx.asm) contain a jump table used by the application to access the Library functions. These files are only used for DFU.

2.7 FILES ORGANIZATION



As you can see on this chart : **USB_Lib.h** and **USB_Def.h** are the only files from the Library that your project needs to refer to.

3 PROJECT

This chapter describes an example of a project architecture. You are not obliged to follow this architecture to use the USB Low-Speed Library.

3.1 DIRECTORIES ARCHITECTURE

The project directories architecture has been simplified. You will now have only the following directories :

```
ProjectName/
   |--- Docs/
   |--- Appli/
   |--- Config/
      |--- Cosmic/
      |--- Hiware/
   |--- Objects/
      |--- Cosmic/
      |--- Hiware/
   |--- ST7USBLS-Library-V4.2x/
```

3.2 DOCS DIRECTORY

This directory contains all documentations related to the project. You can use this directory to put all your project documents.

3.3 APPLI DIRECTORY

All the application-specific files have to be located inside this directory.

Here is a description of the files that are in this directory initially:

File Name	Description
Main.c Main.h	Entry module. Calls the Initialization functions and contains the infinite loop.
Descript.c Descript.h	USB Descriptor files for the application: you have to modify them according to your application (the initial values filled in apply to the Library firmware).

File Name	Description
Int_726x.c	All the interrupt routines for your application have to be in one of these files (depends on the selected microcontroller).
My_Init.c My_Init.h	Your initialisation routines (optional).
User_Var.c User_Var.h	Declaration of all variables related to your application (optional).
USB_App.c USB_App.h	Contains Non Standard USB requests. For example you will find the function “Handle_App_Requests” which interacts with the Library to handle requests that are directed at the application.
HIDlayer.c HIDlayer.h	HID specific functions (like Handle_HID_Requests, Get_Output & Set_Feature).
DFUlayer.c DFUlayer.h	DFU specific functions (like Handle_DFU_Requests).
USB_Opts.h	It is the means by which optional USB library code and functionality is selected for compilation and use by the application. It is also the place for application-specific parameters such as the lengths of Input and Output Reports (but not for descriptors - see Descript.c/.h).
User_Def.h	Declaration of your compilation directives.

3.4 CONFIG/COSMIC DIRECTORY

In this directory are placed all files that are used for the Cosmic compiler only. You will find all the MAK and LKF files for all the ST7 USB Low-Speed microcontrollers. You will find also and the Interrupt Vector files. The content of these files is given as an example and must be updated according to your project.

3.5 CONFIG/HIWARE DIRECTORY

Same as the previous directory excepted that it is for Metrowerks compiler only. You will find the MAK and PRM files for all the ST7 USB Low-Speed microcontrollers. You will find also the Default.env file which contains all the paths description, and the Burner.cmd file used to create a S19 file. Same as above, the content of these files is given as an example and must be updated according to your project.

3.6 OBJECTS/COSMIC AND OBJECTS/HIWARE DIRECTORIES

In this directory you will find all the output files coming from the different tools (compiler, linker, etc...).

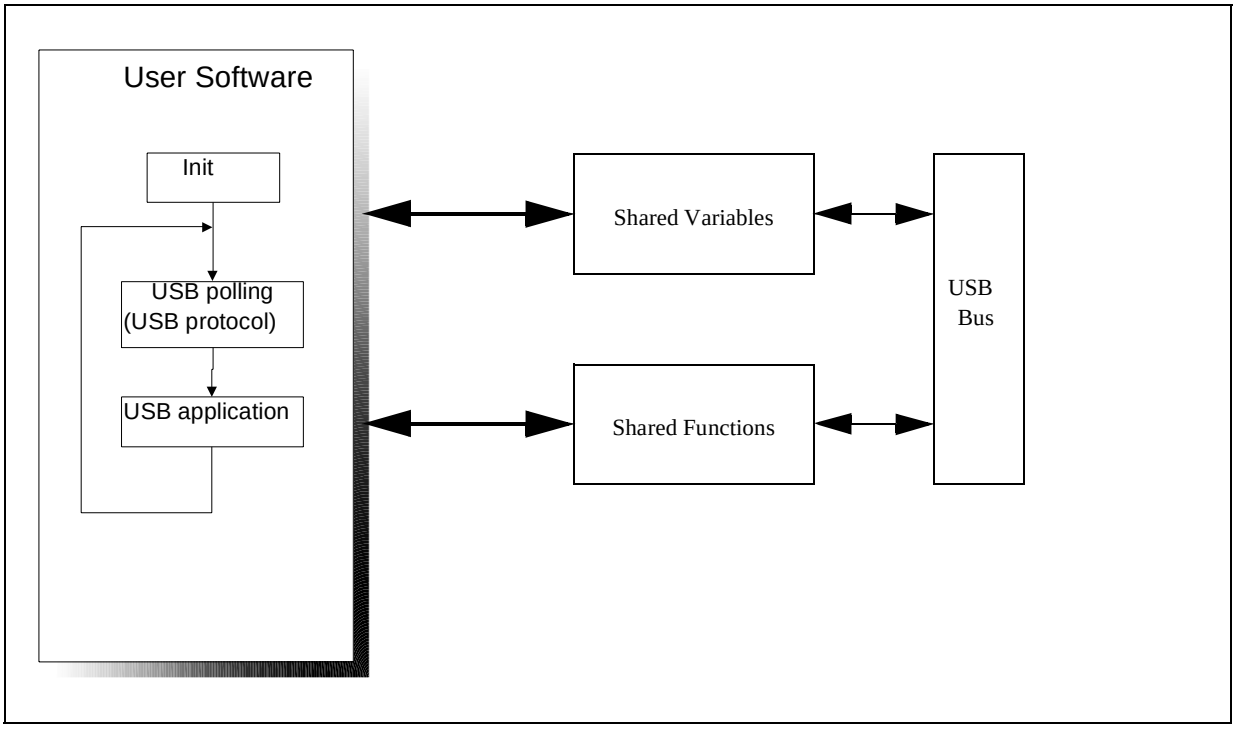
3.7 ST7USBLS-LIBRARY-V4.2X DIRECTORY

This directory contains the Library files described in the beginning of this document.

4 SHARED VARIABLES AND FUNCTIONS

As shown in Figure 1, the USB protocol firmware, (which you do not have to modify) uses a certain number of variables and functions which you can also use in your application source files (Appli directory). These variables and functions are also used in the data transfer functions and interrupt routines.

Figure 1: Interdependancy of USB Variables and Functions



4.1 SHARED VARIABLES

Here is a description of the variables shared between the USB Library and the Application code layer :.

Variable	Declaration/ Aliases	Description
USBLibStatus	USB_Var.c USB_Def.h	Used to exchange USB status information between the Library and the Application. Possible values are : PRODUCT1 (0x01) : always defined PRODUCT2 (0x02) : alternate product DFU_MODE (0x40) : DFU code loader device APP_REQUEST (0x08) : set by Library when a non standard request has been received on EP0 USB_STALL (0x10) : set by Application to stall a non standard request USB_SUSPEND (0x20) : set by Library when suspended mode is requested USB_REMOTE_WAKEUP_ENABLE (0x40) : set by Library when host enables the remote wakeup USB_CONFIGURED (0x80) : set by Library when host configured the peripheral
USBTransferStatus	USB_Var.c USB_Var.h	Specifies the current Control Transfer transaction stages. Possible values are : NO_DATA_STAGE (0x01) : an IN status will follow the setup token DATA_STAGE_IN (0x02) : Transfer with IN data phase DATA_STAGE_OUT (0x04) : Transfer with OUT data phase ONE_MORE (0x08) : Last data IN transfer before status out ADDRESS2SET (0x10) : device address SET_TX0_VALID (0x20) : request to set EP0 Tx buffer to valid SET_RX0_VALID (0x40) : request to set EP0 Rx buffer to valid
USBbmRequestType	USB_Var.c USB_Var.h	Contains the characteristic of the request sent by the host. Refer to USB spec ver 1.1, chap 9.3, page 183 for more informations. Possible values are : RECIPIENT (0x03) REDEVICE (0x00) REINTERFACE (0x01) REENDPOINT (0x02)

Variable	Declaration/ Aliases	Description
USBbRequest	USB_Var.c USB_Def.h	Specific request. Refer to USB spec ver 1.1, chap 9.3, page 183. Possible values are : GET_STATUS, CLEAR_FEATURE_STALL, SET_FEATURE_STALL, SET_ADDRESS, GET_DESCRIPTOR, SET_DESCRIPTOR, GET_CONFIGURATION, SET_CONFIGURATION, GET_INTERFACE, SET_INTERFACE
USBwValue[2]	USB_Var.c	Word-sized field that varies from request. It is used to pass a parameter to the device, specific to the request. Refer to USB spec ver 1.1, chap 9.3, page 183.
USBwIndex	USB_Var.c	Word-sized field that varies from request. Typically used to pass an index or offset. Refer to USB spec ver 1.1, chap 9.3, page 183.
USBwLength	USB_Var.c	Number of bytes to transfer if there is a data stage. Refer to USB spec ver 1.1, chap 9.3, page 183
USBDataXferStatus	USB_Var.c USB_Def.h	Used to send and receive more than 8 bytes for HID requests (i.e. SetFeature, GetFeature, etc.)

4.2 SHARED FUNCTIONS

The table below shows all the functions shared between the Library and the Application. Some functions have been written especially to transfer data over the USB. In order to improve the flexibility and control over the USB flow some functions have been changed compared to the Library Version 3. These new functions will allow also to build a report directly into the End-Point buffer in case the RAM size is critical. The most important functions are described in detail below.

Function	File	Description
Init_USB_HW	USB_Lib.c	Switch ON the USB cell (connect the peripheral to the bus) and initialize Library variables.
Disable_USB_HW	USB_Lib.c	Switch OFF the USB cell (disconnect the peripheral from the bus) and reset Library variables.
Handle_USB_Events	USB_Lib.c	Manage USB flow (interrupts) on EndPoint0 and process standard requests only. Manage also all requests for HID class descriptors (HID, Report or Physical). It's also responsible for STALLing all unsupported and invalid USB requests.
Enable_STATUS_Stage	USB_Lib.c	Allow ST7 to ACK or STALL incoming status stage after completion of a non standard request. This function is called by Application once the APP_REQUESTS processing is done in the Handle_APP_Requests function.

Function	File	Description
Test_EP_Ready	USB_Lib.c	Check if the EndPoint buffer (passed as argument) is ready to be written or read. Return TRUE if the End-Point is ready. For IN direction (ST7 to Host) “ready” means the buffer has been sent and is available for sending new data. For OUT direction (Host to ST7) “ready” means the buffer contains new data to be read.
Set_EP_Ready	USB_Lib.c	This function informs the Library that there is data in the EPxInBuffer ready to be sent (IN direction) or that the Application has read/consumed the data in the EPxOutBuffer (OUT direction). For IN direction (ST7 to Host) update the Transmit byte counter and set EPTx to VALID. For OUT direction (Host to ST7) set the EPRx to VALID and ignore the data length passed as argument.
Write_EP_Buffer	USB_Lib.c	Write the data (pointer passed as argument) into the USB DMA RAM transmit buffer of the selected End-Point.
Read_EP_Buffer	USB_Lib.c	Read data from the USB DMA RAM receive buffer into the data (pointer passed as argument).
Do_USB_RemoteWU	USB_Rwu.c	Send a complete Remote WakeUp signaling sequence to the Host (reset USB_SUSPEND flag). While sending the resume signaling, all interrupts are disable for 10ms (due to K state that must be maintain). If you don’t want all interrupts to be masked for 10ms, when exiting suspend, use the “Start_USB_RemoteWU” and “Stop_USB_RemoteWU” functions instead (see below).
Start_USB_RemoteWU	USB_Rwu.c	Start a Remote WakeUp signaling to the Host. Take care that Application code must call the “Stop_USB_RemoteWU” function after a 10ms delay.
Stop_USB_RemoteWU	USB_Rwu.c	End the Remote WakeUp signaling initiated by the “Start_USB_RemoteWU” function. Reset also the USB_SUSPEND flag.

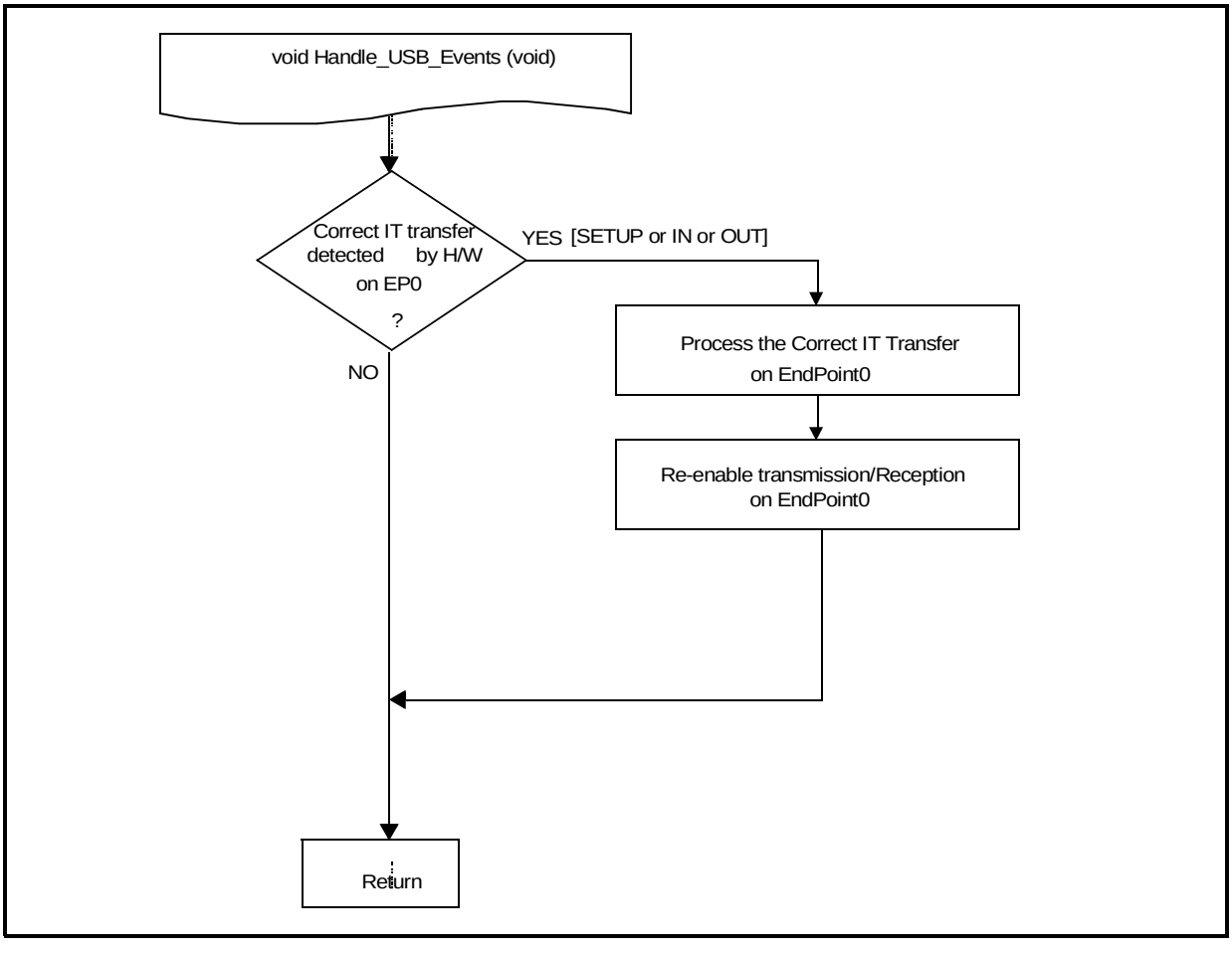
4.2.1 Handle_USB_Events function

Most of the work of the USB Library is performed by this function and the private USB Library functions that it calls. The events to be handled are the assorted USB interrupts that may occur as a result of USB transaction or changes in the state of the bus. The interrupt service routine (INT_Usb) copies the event information to a variable (USBCtrFlag) where the "Handle_USB_Events" function can access it. The event processing occurs when this function is called from the main loop of the application.

This function is responsible for handling all standard ("Chapter 9") requests as well as all requests for HID class descriptors (HID, Report or Physical). It is also responsible for STALLing all unsupported (as indicated via the switches in USB_Opts.h) and invalid USB requests.

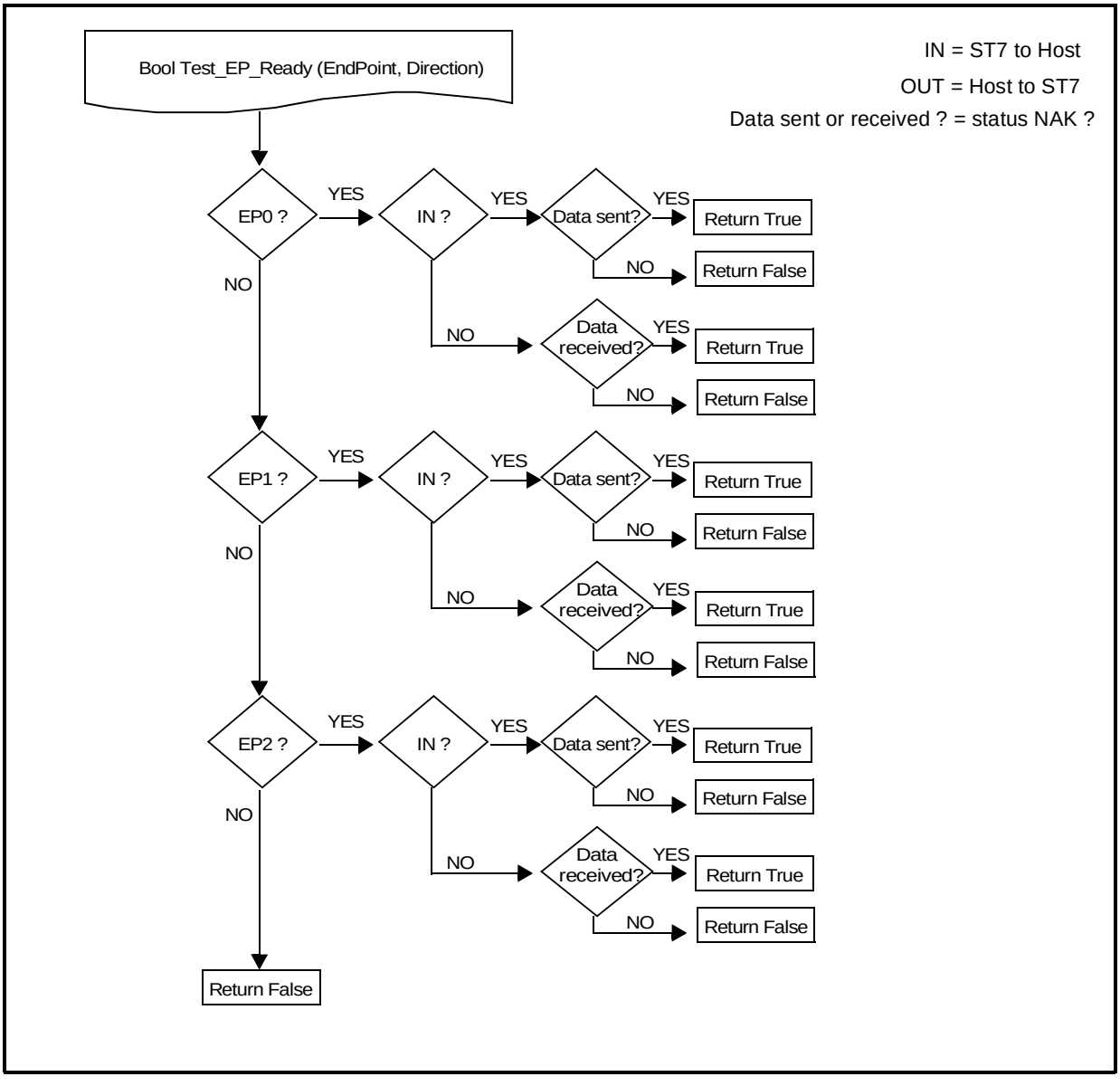
Note: Some USB requests will require attention from the application side of the code (non-descriptor HID requests and all Vendor-specific requests). These requests are addressed in the Application function "Do_App_Requests".

Figure 2: Handle_USB_Events architecture



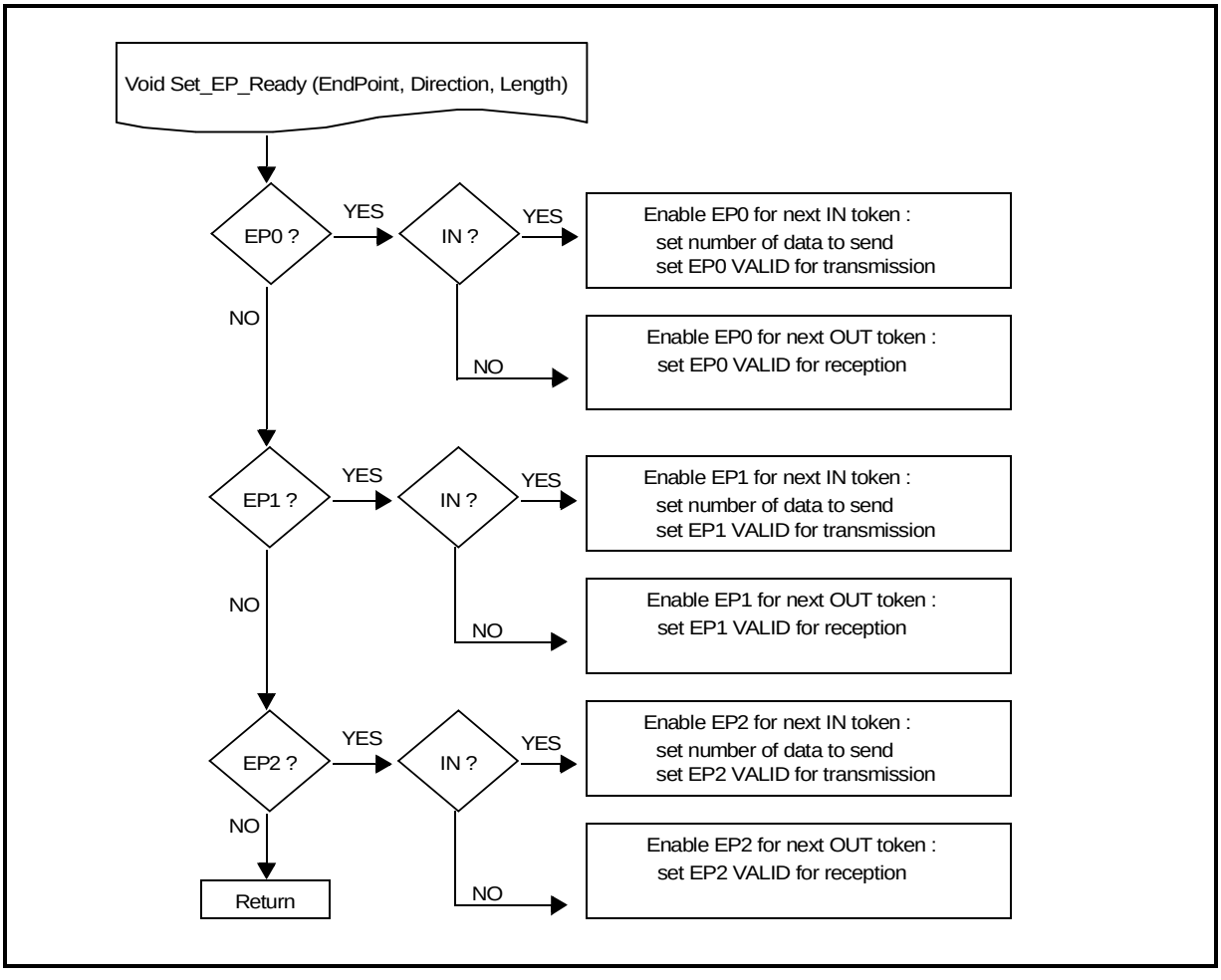
4.2.2 Test_EP_Ready function

Figure 3: Test_EP_Ready architecture



4.2.3 . Set_EP_Ready

Figure 4: Set_EP_Ready architecture



4.3 FUNCTIONS ALREADY AVAILABLE FOR HID CLASS

You will find the following functions in the file “HIDlayer.c” (in “Appli” directory) :

Function	Description
Handle_HID_Requests	Decode Human Interface Device requests.
Get_Output	Build output report (Mandatory if Output usage is defined).
Get_Input	Build input report (Mandatory if Input usage is defined).
Get_Feature	Build feature report (Mandatory if Feature usage is defined).
Set_Feature	Process feature report.
Set_Output	Process output report.

Note: They are given as example only and are not part of the USB core Library.

5 HOW TO START A NEW APPLICATION

1. Make a copy of the complete “EvaluationKit” project. Give an appropriate name to this new directory.

2. Remove all the parts that concern the EvalKit application (modification in Applet.c, Main.c, HIDLayer.c, Int_xxx, ...).

3. Create a new workspace and select the MAK file corresponding to the device you are using.

4. Modify the toolchain configuration files (MAK, PRM, LKF etc...).

5. Modify the file “USB_Opts.h” (in “Appli/Includes” directory) according to your application needs.

6. Modify the files “Descript.c” and “Descript.h” (in “Appli” directory) with your own descriptors. Don’t forget to put your own Vendor ID (assigned by the USB-IF).

7. Add your application in the file “main.c” (in “Appli” directory) inside the “EndPoint 1 & 2 management section”. Build your report_IN and send data or receive data and process the report_OUT). You can also create new files.

8. Process the non-standard requests (Vendor and Class) in the file “USB_App.c” (in “Appli” directory).

9. For HID CLASS requests, use the “HIDlayer.c” and “HIDlayer.h” files (in “Appli” directory).

10. You might want to use as well the predefined modules “User_Var.h” and “User_Def.h” (in “Applis” directory).

6 OTHER DOCUMENTS

For further information on the DFU please refer to the following documents :

- AN1577 “Device Firmware Upgrade (DFU) for ST7 USB devices”