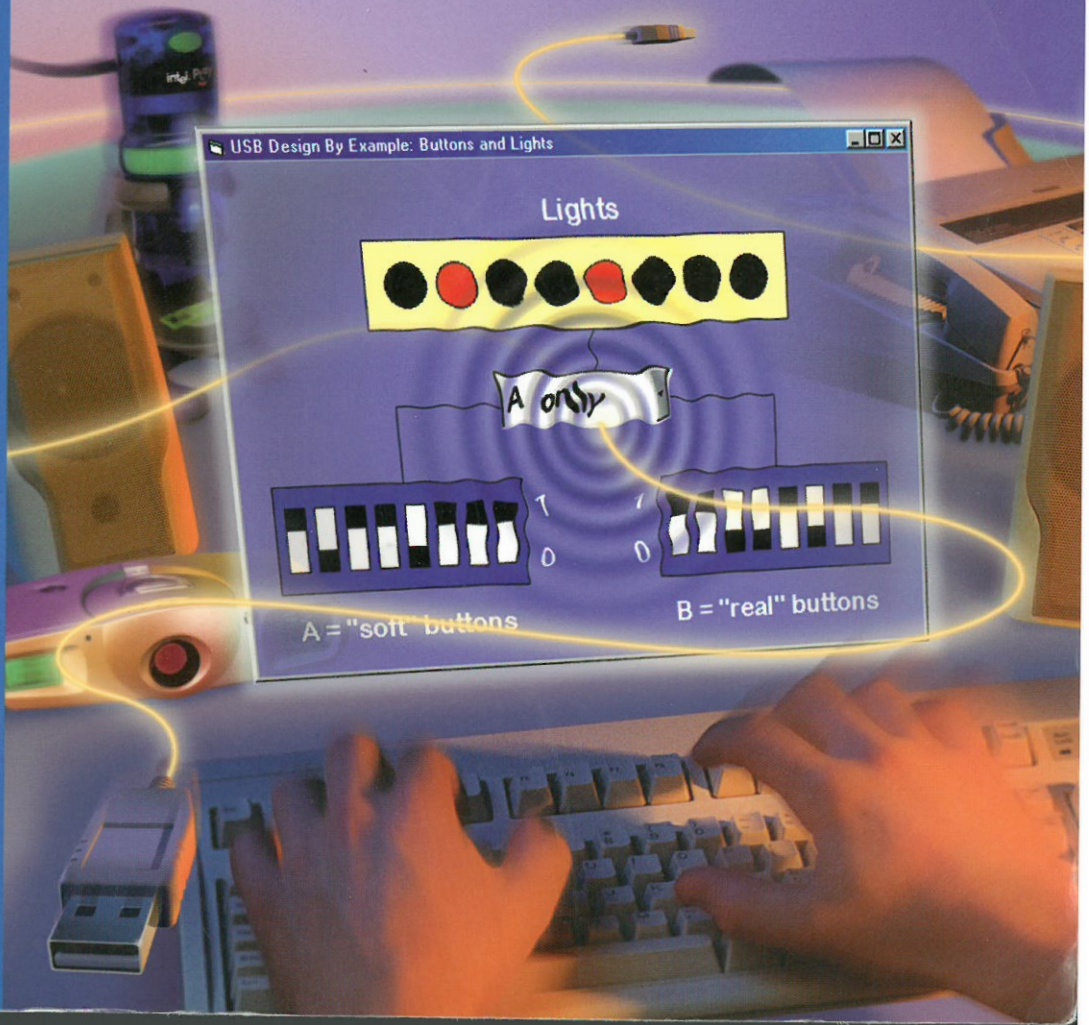
USB Design by Example

A Practical Guide to Building I/O Devices

John Hyde

Intel University Press

The PC Platform Designers' Choice



intel

Publisher: Robert Ipsen (Wiley), Rich Bowles (Intel)
Acquisitions Editor: Cary Sullivan
Editor: Marcia Petty
Assistant Editor: David Spencer
Managing Editor: Frank Grazioli
New Media, Associate Editor: Mike Sosa
Text Design & Composition: Marianne Phelps
Graphic Art: Jerry Heideman (illustrations), Dan Mandish (cover)

Published by John Wiley & Sons, Inc. Published simultaneously in Canada.

This book is printed on acid-free paper. ♾

Library of Congress Cataloging in Publication Data:

Hyde, John, 1952–
USB design by example : a practical guide to building IO devices / John Hyde.
p. cm.
Includes index.
ISBN 0-471-37048-7
1. USB (Computer bus) 2. Computer input-output equipment—Design and construction. I. Title.
TK7895.B87H93 1999
004.6'4—dc21 99-35865
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

The density and complexity of the integrated I/O capability also vary greatly. It is best to design your I/O device first, and then look for the best match of features from a list of available components (see Appendix A for a selection guide).

One device we'll use in several examples is the EZ-USB component from Anchor Chips (Figure 3-12). This component has a unique implementation in that the program memory of the protocol controller is RAM. On power-up, an intelligent SIE holds the protocol controller in reset. The SIE understands the enumeration process and can complete the process without help from the protocol controller. The device driver specified by the EZ-USB device descriptor knows how to do one thing: Download a program into the program RAM and remove the reset from the protocol controller. We thus soft-load a program into the I/O device!

The EZ-USB component then programmatically detaches itself from the hub and reattaches itself with its newly loaded personality. Anchor Chips calls this process "renumeration," and it means that the device doesn't have to be built with a mask ROM, be programmed, or even be flashed. If you think that your I/O device program may change after it ships to users, then software update is as easy as providing a new file on the PC host. No product to recall. No new parts or EPROMs to manufacture and supply. Just ask your users to download an update from the Internet!

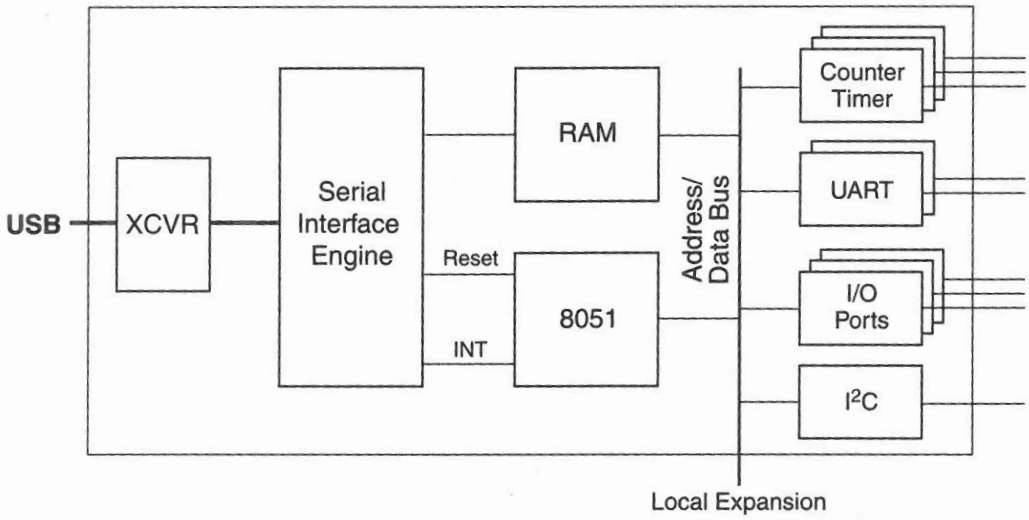


Figure 3-12. EZ-USB component from Anchor Chips

The EZ-USB component is a little more expensive than ROM or EPROM parts because its program in RAM means that the die is larger. For low- and medium-volume projects, however, use of this component could be a cheaper solution overall. Anchor Chips has compatible ROM parts for high volume—these are not downloadable, but you are not penalized for designing a successful product (the part has a lower unit cost but fewer features). Because the many examples in this book have different personalities, I will use the Anchor Chips EZ-USB component for these examples.

COMPLEX I/O DEVICE

This section expands on the minimal descriptors described earlier in the chapter. Edit the descriptors slowly so you can appreciate their building-block nature.

First we'll add a **Strings** descriptor. This has a variable length header followed by variable length string entries (Figure 3-13). The header consists of a **length** and **type** entry (2N+2 and 3) followed by an array (N) of **LanguageIdentifiers**. Each string entry consists of a **length** and **type** entry followed by a unicode **string**. A unicode string uses a word to represent each character and is not NULL-terminated. For more information on unicode, please refer to *The Unicode Standard, Worldwide Character Encoding,* produced by The Unicode Consortium and published by Addison-Wesley, Reading, Massachusetts, U.S.

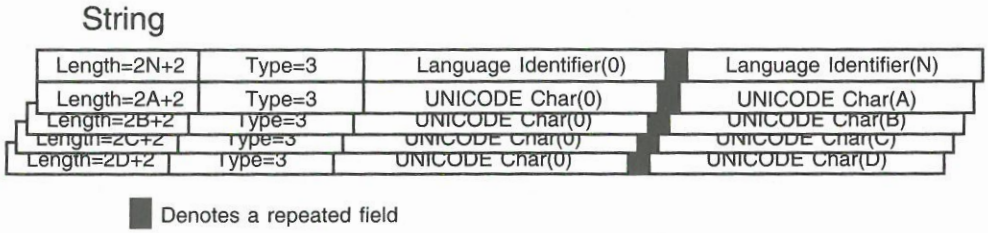


Figure 3-13. Format of a strings descriptor

For the examples here, I use a NULL in the high byte and an ASCII character in the low byte. The order in which the strings are declared will define their **INDEX**—the indexes start from 1 because a 0 is used to define "no string." We can now use useful, human-readable strings in our I/O device and go back to our descriptors and back-fill the string index entries. These strings are helpful during the debug and enumeration phases because they allow Windows to better identify the device (or it uses "unknown device," which is not very helpful).