Dulcimer Reference Manual

Generated by Doxygen 1.5.4

Sat Jul 12 22:27:14 2008

Contents

Chapter 1

Dulcimer

1.1 Introduction

A computer keyboard can be a very personal utensil. Especially if it is an extraordinary well built one, like for example the IBM Model M. The Model M design dates back to 1984, but it still has many fans even nowadays. It came with the usual keyboard connectors. First the old 5-pin one, later a PS/2 plug. Unfortunately is that, at least to my knowledge, they never released a version with USB.

A friend of mine knew that I already had built other USB-devices, and one of them even acted as a keyboard (it isn't really a keyboard, but that's a different story... ;-)). He is a big fan of the Model M, so he asked if I could put new life in one of his old keyboards, which had a broken circuit inside. And this is the result...

1.1.1 Hard- and Software

The main part of a computer keyboard circuit is the key matrix. You can imagine it as a number of keys, placed on a raster of horizontal (rows) and vertical (columns) wires. In the case of a Model M keyboard, we have a matrix of 8x16 lines. Eight columns in 16 rows, or the other way around, depending on how you look at it. Each key is connected to one column and one row. If you press the key, it will connect the column and the row on it's crossing of the lines.

Connected to this matrix is a keyboard controller. That's a chip with a number of I/O-lines to detect the state of the matrix, and on the other side an interface that enables it to talk to the computer. Oh, and not to forget: it also has three output lines to drive the LEDs for Num-, Caps- and Scroll-lock.

What I did in this project is, that I dumped the keyboard controller chip and its circuit, and replaced it by an ATmega32 and my own circuit. The ATmega scans the matrix for keyboard activity, controls the LEDs and talks to the computer.

For further convenience, I added a boot-loader. With that, it is possible to update the keyboard's firmware without disassembling it, and without the need for a dedicated programmer.

1.1.2 Other hardware?

As mentioned, the controller in this project is just connected to an ordinary keyboard matrix. You can find this kind of matrix in all kinds of keyboards, from key-telephones over good old hardware like the Commodore C=64 or the Schneider CPC, keyboards with non-PC-connectors like those made by Sun, to modern hardware that could need a few more features.

Till now, I just made a PCB layout for the IBM Model M, but I intend to modify at least a Sun keyboard. In order to do that, I expect having to refactor the key-scanning, since the key-matrix is not 16x8. The positions of the keys on the matrix will be different, I'll have to re-engineer that. And of course, I'll have to make another PCB.

1.1.3 Features

At the moment, the keyboard should be able to do everything that the average off-the-shelf-keyboard can do. But there are many features that are possible, regarding the fact that the ATmega32 is absolutely bored till now. You can think of 'magic keystrokes' that turn some hidden features on or off, like for example:

- send complete phrases on one keystroke
- 'autofire' feature on keys that don't repeat normally, for example Alt+F4
- change keyboard layout without reconfiguring the computer
- turn bouncing keys on or off, to annoy other people using your computer
- random caps lock function
- use arrow keys as mouse, without having to include a special driver in the OS.

With a little tweaking on the hardware side, there should be even more possibilities:

- turn the oldtimer-keyboard into a supermodern wireless bluetooth one
- implement keylogger-funktionality, using for example an SD-card
- include an USB-hub into the keyboard

If you are just a little like me, it won't take you much brainstorming to come up with own useful – or even better: useless – ideas. ;-)

1.2 Building and installing

Both, the bootloader and firmware are simply built with "make". You may need to customize both makefiles to fit to your system. If you don't want to add new features, you don't need to build the software yourself. You can use the hex-files included in this package.

1.2.1 Bootloader

I used the USBaspLoader from Objective Development, the same guys that wrote the AVR-USB-driver: http://www.obdev.at/products/avrusb/usbasploader.html

The reason why I chose this over some other available USB-bootloaders is, that this one emulates a common ISP-programmer that is supported by avrdude. In this way, the same program can be used to program the chip that is used without a bootloader.

To prepare the ATmega32, you have to connect it to your computer with the ISP-programmer of your choice and modify the makefile according to that. Then you enter the bootloader-directory and enter the following line:

```
make fuse && make flash && make lock
```

With 'fuse' you prepare the fuse-bits of your AVR, 'flash' transfers the bootloader to the device and 'lock' prevents you from overwriting the bootloader. Don't fear the locking: you can always reset it with your ordinary programmer. In fact, it is disabled in the moment you use your ordinary programmer to reflash the device, even without any special parameters. The locking only affects the bootloader behavior.

Afterwards you can put the programmer back into the toolbox, you won't need it from here on.

When you plug in the device while holding the minus-key on the number-keypad pressed, the keyboard indicates that it would like to get a new firmware by showing a running light on the LEDs. That firmware will be flashed over the normal USB-cable that the keyboard is connected with.

1.2.2 Firmware

If you intend to recompile the firmware yourself, you will need avr-gcc and avr-libc (a C-library for the AVR controller). Please read the instructions at http://www.nongnu.org/avr-libc/user-manual/install_tools.html for how to install the GNU toolchain (avr-gcc, assembler, linker etc.) and avr-libc.

Once you have the GNU toolchain for AVR microcontrollers installed, you can run "make" in the subdirectory "firmware".

Afterwards – or if you decided not to compile the firmware yourself – you can flash it to the device:

```
make program
```

Remember that you have to start the bootloader at first: unplug the keyboard, hold the minus-key on the number-keypad pressed and replug it. If the modified keyboard is the only one within reach: good luck! ;-)

1.3 Usage

Connect the keyboard to the USB-port. All LED should flash up to indicate that the device is initialized.

Then you can use the keyboard as always. If additional features get implemented, you will be able to use them in their respective ways.

1.4 Drawbacks

I don't know if and how keyboard manufacturers face the problem of ghost keys, I didn't take special measurements for this. I hope that the engineers at IBM distributed the keys on the matrix in a way that minimizes this problem. Don't misunderstand: I haven't experienced that on this keyboard, but I know that it's a common problem on key-matrixes.

1.5 Files in the distribution

- *Readme.txt:* Documentation, created from the htmldoc-directory.
- *firmware:* Source code of the controller firmware.
- *firmware/usbdrv*: USB driver – See Readme.txt in this directory for info.
- *bootloader:* The USBaspLoader, properly configured for this project. I only modified the bootloaderconfig.h and the Makefile.

- *USBaspLoader.2008-02-05.tar.gz*: The unmodified bootloader sources, for reference.
- *circuit:* Circuit diagrams in PDF and KiCAD format. KiCAD is a free schematic- and layout-tool, you can learn more about it at its homepage: http://www.lis.inpg.fr/realise_au_-lis/kicad/
- *License.txt:* Public license for all contents of this project, except for the USB driver. Look in firmware/usbdrv/License.txt for further info.
- *Changelog.txt:* Logfile documenting changes in soft-, firm- and hardware.
- *refman.pdf:* Full documentation of the software.

1.6 Thanks!

I'd like to thank **Objective Development** for the possibility to use their driver for my project. In fact, this project wouldn't exist without the driver.

And of course I'd like to thank that friend of mine – I doubt that he'd like to read his name in this place, I'll put it in if he wants me to – that gave me the idea for this project.

1.7 About the license

Chapter 2

Dulcimer Data Structure Index

2.1 Dulcimer Data Structures

Here are the data structures with brief descriptions:

Chapter 3

Dulcimer File Index

3.1 Dulcimer File List

Here is a list of all files with brief descriptions:

Chapter 4

Dulcimer Data Structure Documentation

4.1 Key Struct Reference

This structure can be used as a container for a single 'key'.

Data Fields

- uint8_t mode
- uint8_t key

4.1.1 Detailed Description

This structure can be used as a container for a single 'key'.

It consists of the key-code and the modifier-code.

Definition at line 482 of file main.c.

4.1.2 Field Documentation

4.1.2.1 uint8_t Key::mode

Definition at line 483 of file main.c.

Referenced by charToKey(), and sendKey().

4.1.2.2 uint8_t Key::key

Definition at line 484 of file main.c.

Referenced by charToKey(), and sendKey().

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

- firmware/main.c

Chapter 5

Dulcimer File Documentation

5.1 bootloader/bootloaderconfig.h File Reference

This file (together with some settings in Makefile) configures the boot loader according to the hardware.

Defines

- #define USB_CFG_IOPORTNAME D

  This is the port where the USB bus is connected.

- #define USB_CFG_DMINUS_BIT 0

  This is the bit number in USB_CFG_IOPORT where the USB D- line is connected.

- #define USB_CFG_DPLUS_BIT 2

  This is the bit number in USB_CFG_IOPORT where the USB D+ line is connected.

- #define USB_CFG_CLOCK_KHZ (F_CPU/1000)

  Clock rate of the AVR in MHz.

- #define HAVE_EEPROM_PAGED_ACCESS 1

  If HAVE_EEPROM_PAGED_ACCESS is defined to 1, page mode access to EEPROM is compiled in.

- #define HAVE_EEPROM_BYTE_ACCESS 1

  If HAVE_EEPROM_BYTE_ACCESS is defined to 1, byte mode access to EEPROM is compiled in.

- #define BOOTLOADER_CAN_EXIT 1

  If this macro is defined to 1, the boot loader will exit shortly after the programmer closes the connection to the device.

- #define SIGNATURE_BYTES 0x1e, 0x95, 0x02, 0

  This macro defines the signature bytes returned by the emulated USBasp to the programmer software.

Variables

- uint8_t ledcounter = 0

  counter used to set the speed of the running light

- uint8_t ledstate = 0

  state of the running light

5.1.1 Detailed Description

This file (together with some settings in Makefile) configures the boot loader according to the hardware.

This file contains (besides the hardware configuration normally found in usbconfig.h) two functions or macros: bootLoaderInit() and bootLoaderCondition(). Whether you implement them as macros or as static inline functions is up to you, decide based on code size and convenience.

bootLoaderInit() is called as one of the first actions after reset. It should be a minimum initialization of the hardware so that the boot loader condition can be read. This will usually consist of activating a pull-up resistor for an external jumper which selects boot loader mode.

bootLoaderCondition() is called immediately after initialization and in each main loop iteration. If it returns TRUE, the boot loader will be active. If it returns FALSE, the boot loader jumps to address 0 (the loaded application) immediately.

For compatibility with Thomas Fischl's avrusbboot, we also support the macro names BOOTLOADER_INIT and BOOTLOADER_CONDITION for this functionality. If these macros are defined, the boot loader usees them.

Author:

Ronald Schaten <ronald@schatenseite.de>

Version:

Id

bootloaderconfig.h,v 1.1 2008-07-09 20:47:11 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file bootloaderconfig.h.

5.1.2 Define Documentation

5.1.2.1 #define BOOTLOADER_CAN_EXIT 1

If this macro is defined to 1, the boot loader will exit shortly after the programmer closes the connection to the device.

Costs ~36 bytes.

Definition at line 91 of file bootloaderconfig.h.

5.1.2.2 #define HAVE_EEPROM_BYTE_ACCESS 1

If HAVE_EEPROM_BYTE_ACCESS is defined to 1, byte mode access to EEPROM is compiled in.

Byte mode is only used if the device (as identified by its signature) does not support page mode for EEPROM. It is required for accessing the EEPROM on the ATMega8. Costs ∼54 bytes.

Definition at line 87 of file bootloaderconfig.h.

5.1.2.3 #define HAVE_EEPROM_PAGED_ACCESS 1

If HAVE_EEPROM_PAGED_ACCESS is defined to 1, page mode access to EEPROM is compiled in.

Whether page mode or byte mode access is used by AVRDUDE depends on the target device. Page mode is only used if the device supports it, e.g. for the ATMega88, 168 etc. You can save quite a bit of memory by disabling page mode EEPROM access. Costs ∼ 138 bytes.

Definition at line 81 of file bootloaderconfig.h.

5.1.2.4 #define SIGNATURE_BYTES 0x1e, 0x95, 0x02, 0

This macro defines the signature bytes returned by the emulated USBasp to the programmer software.

They should match the actual device at least in memory size and features. If you don't define this, values for ATMega8, ATMega88, ATMega168 and ATMega328 are guessed correctly.

Definition at line 97 of file bootloaderconfig.h.

5.1.2.5 #define USB_CFG_CLOCK_KHZ (F_CPU/1000)

Clock rate of the AVR in MHz.

Legal values are 12000, 16000 or 16500. The 16.5 MHz version of the code requires no crystal, it tolerates +/- 1% deviation from the nominal frequency. All other rates require a precision of 2000 ppm and thus a crystal! Default if not specified: 12 MHz

Definition at line 55 of file bootloaderconfig.h.

5.1.2.6 #define USB_CFG_DMINUS_BIT 0

This is the bit number in USB_CFG_IOPORT where the USB D- line is connected.

This may be any bit in the port.

Definition at line 43 of file bootloaderconfig.h.

5.1.2.7 #define USB_CFG_DPLUS_BIT 2

This is the bit number in USB_CFG_IOPORT where the USB D+ line is connected.

This may be any bit in the port. Please note that D+ must also be connected to interrupt pin INT0!

Definition at line 48 of file bootloaderconfig.h.

5.1.2.8 #define USB_CFG_IOPORTNAME D

This is the port where the USB bus is connected.

When you configure it to "B", the registers PORTB, PINB and DDRB will be used.

Definition at line 39 of file bootloaderconfig.h.

5.1.3 Variable Documentation

5.1.3.1 uint8_t ledcounter = 0

counter used to set the speed of the running light

Definition at line 118 of file bootloaderconfig.h.

5.1.3.2 uint8_t ledstate = 0

state of the running light

Definition at line 119 of file bootloaderconfig.h.

5.2 bootloader/main.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <avr/boot.h>
#include <avr/eeprom.h>
#include <util/delay.h>
#include <string.h>
#include "bootloaderconfig.h"
#include "usbdrv/usbdrv.c"
```

Defines

- #define USBASP_FUNC_CONNECT 1
- #define USBASP_FUNC_DISCONNECT 2
- #define USBASP_FUNC_TRANSMIT 3
- #define USBASP_FUNC_READFLASH 4
- #define USBASP_FUNC_ENABLEPROG 5
- #define USBASP_FUNC_WRITEFLASH 6
- #define USBASP_FUNC_READEEPROM 7
- #define USBASP_FUNC_WRITEEEPROM 8
- #define USBASP_FUNC_SETLONGADDRESS 9
- #define ulong unsigned long
- #define uint unsigned int
- #define GICR MCUCR
- #define CURRENT_ADDRESS currentAddress.w[0]

Functions

- uchar usbFunctionSetup (uchar data[8])
- uchar usbFunctionWrite (uchar *data, uchar len)
- uchar usbFunctionRead (uchar *data, uchar len)
- int main (void)

Variables

- longConverter_t

5.2.1 Define Documentation

5.2.1.1 #define CURRENT_ADDRESS currentAddress.w[0]

Definition at line 106 of file main.c.

5.2.1.2 #define GICR MCUCR

5.2.1.3 #define uint unsigned int

5.2.1.4 #define ulong unsigned long

5.2.1.5 #define USBASP_FUNC_CONNECT 1

5.2.1.6 #define USBASP_FUNC_DISCONNECT 2

5.2.1.7 #define USBASP_FUNC_ENABLEPROG 5

5.2.1.8 #define USBASP_FUNC_READEEPROM 7

5.2.1.9 #define USBASP_FUNC_READFLASH 4

5.2.1.10 #define USBASP_FUNC_SETLONGADDRESS 9

5.2.1.11 #define USBASP_FUNC_TRANSMIT 3

5.2.1.12 #define USBASP_FUNC_WRITEEEPROM 8

5.2.1.13 #define USBASP_FUNC_WRITEFLASH 6

5.2.2 Function Documentation

5.2.2.1 int main (void)

Definition at line 265 of file main.c.

5.2.2.2 uchar usbFunctionRead (uchar * *data*, uchar *len*)

Definition at line 234 of file main.c.

5.2.2.3 uchar usbFunctionSetup (uchar *data*[8])

Definition at line 130 of file main.c.

5.2.2.4 uchar usbFunctionWrite (uchar * *data*, uchar *len*)

Definition at line 184 of file main.c.

5.2.3 Variable Documentation

5.2.3.1 longConverter_t

Definition at line 78 of file main.c.

5.3 firmware/main.c File Reference

Main functions for USB-keyboard.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <string.h>
#include <stdio.h>
#include "usbdrv.h"
#include "keycodes.h"
```

Data Structures

- struct Key

  This structure can be used as a container for a single 'key'.

Defines

- #define F_CPU 12000000L

  we use a 12MHz crystal

- #define PORTCOLUMNS PORTB

  port on which we read the state of the columns

- #define PINCOLUMNS PINB

  port on which we read the state of the columns

- #define DDRCOLUMNS DDRB

  port on which we read the state of the columns

- #define PORTROWS1 PORTA

  first port connected to the matrix rows

- #define PINROWS1 PINA

  first port connected to the matrix rows

- #define DDRROWS1 DDRA

  first port connected to the matrix rows

- #define PORTROWS2 PORTC

  second port connected to the matrix rows

- #define PINROWS2 PINC

second port connected to the matrix rows

- #define DDRROWS2 DDRC
  second port connected to the matrix rows
- #define PORTLEDS PORTD
  port on which the LEDs are connected
- #define PINLEDS PIND
  port on which the LEDs are connected
- #define DDRLEDS DDRD
  port on which the LEDs are connected
- #define LEDSCROLL PIND4
  address of the scroll-lock LED
- #define LEDCAPS PIND5
  address of the caps-lock LED
- #define LEDNUM PIND6
  address of the num-lock LED
- #define PORTJUMPERS PORTD
  port for additional jumpers
- #define PINJUMPERS PIND
  port for additional jumpers
- #define DDRJUMPERS DDRD
  port for additional jumpers
- #define JUMPER0 PD1
  address for jumper 0
- #define JUMPER1 PD3
  address for jumper 1
- #define JUMPER2 PD7
  address for jumper 2
- #define LED_NUM 0x01
  num LED on a boot-protocol keyboard
- #define LED_CAPS 0x02
  caps LED on a boot-protocol keyboard
- #define LED_SCROLL 0x04
  scroll LED on a boot-protocol keyboard

- #define LED_COMPOSE 0x08

  compose LED on a boot-protocol keyboard

- #define LED_KANA 0x10

  kana LED on a boot-protocol keyboard

Functions

- uint8_t usbFunctionSetup (uint8_t data[8])

  This function is called whenever we receive a setup request via USB.

- uint8_t usbFunctionWrite (uchar *data, uchar len)

  The write function is called when LEDs should be set.

- void usbSendReport (uint8_t mode, uint8_t key)

  Send a single report to the computer.

- Key charToKey (char character)

  Convert an ASCII-character to the corresponding key-code and modifier-code combination.

- void sendKey (Key keytosend)

  Send a key to the computer, followed by the release of all keys.

- void sendString (char *string)

  Send a string to the computer.

- void printMatrix (void)

  Print the current state of the keyboard in a readable form.

- uint8_t scankeys (void)

  Scan and debounce keypresses.

- int main (void)

  Main function, containing the main loop that manages timer- and USB-functionality.

Variables

- uint8_t expectReport = 0

  flag to indicate if we expect an USB-report

- uint8_t LEDstate = 0

  current state of the LEDs

- char PROGMEM usbHidReportDescriptor [USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH]

  USB report descriptor (length is defined in usbconfig.h).

- uint8_t curmatrix [16]

  contains current state of the keyboard

- const uint8_t PROGMEM keymatrix [16][8]

  The keymatrix-array contains positions of keys in the matrix.

- const uint8_t PROGMEM modmatrix [16][8]

  The modmatrix-array contains positions of the modifier-keys in the matrix.

5.3.1 Detailed Description

Main functions for USB-keyboard.

Author:

Ronald Schaten <ronald@schatenseite.de>

Version:

Id

main.c,v 1.5 2008-07-12 21:05:24 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file main.c.

5.3.2 Define Documentation

5.3.2.1 #define DDRCOLUMNS DDRB

port on which we read the state of the columns

Definition at line 225 of file main.c.

5.3.2.2 #define DDRJUMPERS DDRD

port for additional jumpers

Definition at line 242 of file main.c.

5.3.2.3 #define DDRLEDS DDRD

port on which the LEDs are connected

Definition at line 235 of file main.c.

5.3.2.4 #define DDRROWS1 DDRA

first port connected to the matrix rows

Definition at line 228 of file main.c.

Referenced by scankeys().

5.3.2.5 #define DDRROWS2 DDRC

second port connected to the matrix rows

Definition at line 231 of file main.c.

Referenced by scankeys().

5.3.2.6 #define F_CPU 12000000L

we use a 12MHz crystal

Definition at line 207 of file main.c.

5.3.2.7 #define JUMPER0 PD1

address for jumper 0

Definition at line 243 of file main.c.

5.3.2.8 #define JUMPER1 PD3

address for jumper 1

Definition at line 244 of file main.c.

5.3.2.9 #define JUMPER2 PD7

address for jumper 2

Definition at line 245 of file main.c.

5.3.2.10 #define LED_CAPS 0x02

caps LED on a boot-protocol keyboard

Definition at line 291 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.11 #define LED_COMPOSE 0x08

compose LED on a boot-protocol keyboard

Definition at line 293 of file main.c.

5.3.2.12 #define LED_KANA 0x10

kana LED on a boot-protocol keyboard

Definition at line 294 of file main.c.

5.3.2.13 #define LED_NUM 0x01

num LED on a boot-protocol keyboard

Definition at line 290 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.14 #define LED_SCROLL 0x04

scroll LED on a boot-protocol keyboard

Definition at line 292 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.15 #define LEDCAPS PIND5

address of the caps-lock LED

Definition at line 237 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.16 #define LEDNUM PIND6

address of the num-lock LED

Definition at line 238 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.17 #define LEDSCROLL PIND4

address of the scroll-lock LED

Definition at line 236 of file main.c.

Referenced by usbFunctionWrite().

5.3.2.18 #define PINCOLUMNS PINB

port on which we read the state of the columns

Definition at line 224 of file main.c.

Referenced by scankeys().

5.3.2.19 #define PINJUMPERS PIND

port for additional jumpers
Definition at line 241 of file main.c.

5.3.2.20 #define PINLEDS PIND

port on which the LEDs are connected
Definition at line 234 of file main.c.

5.3.2.21 #define PINROWS1 PINA

first port connected to the matrix rows
Definition at line 227 of file main.c.

5.3.2.22 #define PINROWS2 PINC

second port connected to the matrix rows
Definition at line 230 of file main.c.

5.3.2.23 #define PORTCOLUMNS PORTB

port on which we read the state of the columns
Definition at line 223 of file main.c.

5.3.2.24 #define PORTJUMPERS PORTD

port for additional jumpers
Definition at line 240 of file main.c.

5.3.2.25 #define PORTLEDS PORTD

port on which the LEDs are connected
Definition at line 233 of file main.c.
Referenced by usbFunctionWrite().

5.3.2.26 #define PORTROWS1 PORTA

first port connected to the matrix rows
Definition at line 226 of file main.c.
Referenced by scankeys().

5.3.2.27 #define PORTROWS2 PORTC

second port connected to the matrix rows

Definition at line 229 of file main.c.

Referenced by scankeys().

5.3.3 Function Documentation

5.3.3.1 Key charToKey (char *character*)

Convert an ASCII-character to the corresponding key-code and modifier-code combination.

character ASCII-character to convert

Returns:

structure containing the combination

Definition at line 493 of file main.c.

References Key::key, KEY_0, KEY_1, KEY_4, KEY_5, KEY_6, KEY_7, KEY_8, KEY_9, KEY_apostroph, KEY_backslash, KEY_comma, KEY_dot, KEY_equals, KEY_grave, KEY_hash, KEY_lbracket, KEY_minus, KEY_rbracket, KEY_Reserved, KEY_semicolon, KEY_slash, KEY_Spacebar, MOD_NONE, MOD_SHIFT_LEFT, and Key::mode.

Referenced by sendString().

5.3.3.2 int main (void)

Main function, containing the main loop that manages timer- and USB-functionality.

/return the obligatory integer that nobody cares about...

Definition at line 755 of file main.c.

References scankeys().

5.3.3.3 void printMatrix (void)

Print the current state of the keyboard in a readable form.

This function is used for debug-purposes only.

Definition at line 642 of file main.c.

References curmatrix, and sendString().

5.3.3.4 uint8_t scankeys (void)

Scan and debounce keypresses.

This is the main worker function for normal keyboard operation, the code contains lot of comments. Basically, it first scans the keyboard state. If a change is detected, it initializes a counter that is decreased each time this function is called. If the counter reaches 1, that means that the same scan result has been scanned ten times in a row, so we can be pretty sure that the keys are in a certain state (as in: not bouncing). Then,

the codes for keys and modifiers are searched from the two arrays, the USB-message to send the state is prepared. The return value of this function indicates if the message has to be sent.

Returns:

flag to indicate whether something has changed

Definition at line 679 of file main.c.

References curmatrix, DDRROWS1, DDRROWS2, KEY_ErrorRollOver, KEY_Reserved, keymatrix, MOD_NONE, modmatrix, PINCOLUMNS, PORTROWS1, and PORTROWS2.

Referenced by main().

5.3.3.5 void sendKey (Key *keytosend*)

Send a key to the computer, followed by the release of all keys.

This can be used repetitively to send a string.

Parameters:

keytosend key structure to send

Definition at line 621 of file main.c.

References Key::key, Key::mode, and usbSendReport().

Referenced by sendString().

5.3.3.6 void sendString (char * *string*)

Send a string to the computer.

This function converts each character of an ASCII-string to a key-structure and uses sendKey() to send it.

Parameters:

string string to send

Definition at line 631 of file main.c.

References charToKey(), and sendKey().

Referenced by printMatrix().

5.3.3.7 uint8_t usbFunctionSetup (uint8_t *data*[8])

This function is called whenever we receive a setup request via USB.

Parameters:

data[8] eight bytes of data we received

Returns:

number of bytes to use, or 0xff if usbFunctionWrite() should be called

Definition at line 343 of file main.c.

References expectReport.

5.3.3.8 uint8_t usbFunctionWrite (uchar * *data*, uchar *len*)

The write function is called when LEDs should be set.

Normally, we get only one byte that contains info about the LED states.

Parameters:

data pointer to received data

len number ob bytes received

Returns:

0x01

Definition at line 384 of file main.c.

References expectReport, LED_CAPS, LED_NUM, LED_SCROLL, LEDCAPS, LEDNUM, LEDSCROLL, LEDstate, and PORTLEDS.

5.3.3.9 void usbSendReport (uint8_t *mode*, uint8_t *key*)

Send a single report to the computer.

This function is not used during normal typing, it is only used to send non-pressed keys to simulate input.

Parameters:

mode modifier-byte

key key-code

Definition at line 413 of file main.c.

Referenced by sendKey().

5.3.4 Variable Documentation

5.3.4.1 uint8_t curmatrix[16]

contains current state of the keyboard

Definition at line 424 of file main.c.

Referenced by printMatrix(), and scankeys().

5.3.4.2 uint8_t expectReport = 0

flag to indicate if we expect an USB-report

Definition at line 288 of file main.c.

Referenced by usbFunctionSetup(), and usbFunctionWrite().

5.3.4.3 const uint8_t PROGMEM keymatrix[16][8]

Initial value:

```
{
    {KEY_Reserved,  KEY_Reserved, KEY_Reserved,      KEY_Reserved,    KEY_Reserved,    KEY_Reserved,   KEY_F
    {KEY_Reserved,  KEY_Reserved, KEY_Reserved,      KEY_Reserved,    KEY_Reserved,    KEY_Reserved,   KEY_F
    {KEY_ESCAPE,    KEY_Tab,      KEY_grave,         KEY_1,           KEY_Q,           KEY_A,          KEY_Z
    {KEY_Euro,      KEY_capslock, KEY_F1,            KEY_2,           KEY_W,           KEY_S,          KEY_X
    {KEY_F4,        KEY_F3,       KEY_F2,            KEY_3,           KEY_E,           KEY_D,          KEY_C
    {KEY_G,         KEY_T,        KEY_5,             KEY_4,           KEY_R,           KEY_F,          KEY_V
    {KEY_F5,        KEY_DELETE,   KEY_F9,            KEY_F10,         KEY_Reserved,    KEY_Reserved,   KEY_F
    {KEY_H,         KEY_Y,        KEY_6,             KEY_7,           KEY_U,           KEY_J,          KEY_M
    {KEY_F6,        KEY_rbracket, KEY_equals,        KEY_8,           KEY_I,           KEY_K,          KEY_c
    {KEY_Reserved,  KEY_F7,       KEY_F8,            KEY_9,           KEY_O,           KEY_L,          KEY_c
    {KEY_apostroph, KEY_lbracket, KEY_minus,         KEY_0,           KEY_P,           KEY_semicolon,  KEY_h
    {KEY_Reserved,  KEY_KP4,      KEY_DeleteForward, KEY_F11,         KEY_KP7,         KEY_KP1,        KEY_N
    {KEY_KP0,       KEY_KP5,      KEY_Insert,        KEY_F12,         KEY_KP8,         KEY_KP2,        KEY_K
    {KEY_KPcomma,   KEY_KP6,      KEY_PageUp,        KEY_PageDown,    KEY_KP9,         KEY_KP3,        KEY_K
    {KEY_UpArrow,   KEY_Reserved, KEY_Home,          KEY_End,         KEY_KPplus,      KEY_KPenter,    KEY_P
    {KEY_Reserved,  KEY_Reserved, KEY_Reserved,      KEY_PrintScreen, KEY_ScrollLock,  KEY_Reserved,   KEY_F
}
```

The keymatrix-array contains positions of keys in the matrix.

Here you can see which row is connected to which column when a key is pressed. This array probably has to be modified if this firmware is ported to a different keyboard.

See also:

modmatrix

Definition at line 433 of file main.c.

Referenced by scankeys().

5.3.4.4 uint8_t LEDstate = 0

current state of the LEDs

Definition at line 295 of file main.c.

Referenced by usbFunctionWrite().

5.3.4.5 const uint8_t PROGMEM modmatrix[16][8]

Initial value:

```
{
    {MOD_NONE,      MOD_NONE,        MOD_CONTROL_LEFT, MOD_NONE, MOD_NONE, MOD_NONE, MOD_CONTROL_RIGHT, MOD_
    {MOD_NONE,      MOD_SHIFT_LEFT,  MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_SHIFT_RIGHT,   MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,      MOD_NONE,        MOD_NONE,         MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
```

```
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_NONE,     MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
    {MOD_ALT_LEFT, MOD_NONE,       MOD_NONE,        MOD_NONE, MOD_NONE, MOD_NONE, MOD_NONE,          MOD_
}
```

The modmatrix-array contains positions of the modifier-keys in the matrix.

It is built in the same way as the keymatrix-array.

See also:

keymatrix

Definition at line 458 of file main.c.

Referenced by scankeys().

5.3.4.6 char PROGMEM usbHidReportDescriptor[USB_CFG_HID_REPORT_DESCRIPTOR_-LENGTH]

USB report descriptor (length is defined in usbconfig.h).

The report descriptor has been created with usb.org's "HID Descriptor Tool" which can be downloaded from `http://www.usb.org/developers/hidpage/` (it's an .exe, but it even runs under Wine).

Definition at line 302 of file main.c.

5.4 bootloader/usbconfig.h File Reference

```
#include "bootloaderconfig.h"
```

Defines

- #define USB_PUBLIC static
- #define USB_CFG_HAVE_INTRIN_ENDPOINT 0
- #define USB_CFG_HAVE_INTRIN_ENDPOINT3 0
- #define USB_CFG_IMPLEMENT_HALT 0
- #define USB_CFG_INTR_POLL_INTERVAL 200
- #define USB_CFG_IS_SELF_POWERED 0
- #define USB_CFG_MAX_BUS_POWER 100
- #define USB_CFG_IMPLEMENT_FN_WRITE 1
- #define USB_CFG_IMPLEMENT_FN_READ 1
- #define USB_CFG_IMPLEMENT_FN_WRITEOUT 0
- #define USB_CFG_HAVE_FLOWCONTROL 0
- #define USB_CFG_VENDOR_ID 0xc0, 0x16
- #define USB_CFG_DEVICE_ID 0xdc, 0x05
- #define USB_CFG_DEVICE_VERSION 0x02, 0x01
- #define USB_CFG_VENDOR_NAME 'w', 'w', 'w', '.', 'f', 'i', 's', 'c', 'h', 'l', '.', 'd', 'e'
- #define USB_CFG_VENDOR_NAME_LEN 13
- #define USB_CFG_DEVICE_NAME 'U', 'S', 'B', 'a', 's', 'p'
- #define USB_CFG_DEVICE_NAME_LEN 6
- #define USB_CFG_DEVICE_CLASS 0xff
- #define USB_CFG_DEVICE_SUBCLASS 0
- #define USB_CFG_INTERFACE_CLASS 0
- #define USB_CFG_INTERFACE_SUBCLASS 0
- #define USB_CFG_INTERFACE_PROTOCOL 0
- #define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH 0
- #define USB_CFG_DESCR_PROPS_DEVICE 0
- #define USB_CFG_DESCR_PROPS_CONFIGURATION 0
- #define USB_CFG_DESCR_PROPS_STRINGS 0
- #define USB_CFG_DESCR_PROPS_STRING_0 0
- #define USB_CFG_DESCR_PROPS_STRING_VENDOR 0
- #define USB_CFG_DESCR_PROPS_STRING_PRODUCT 0
- #define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER 0
- #define USB_CFG_DESCR_PROPS_HID 0
- #define USB_CFG_DESCR_PROPS_HID_REPORT 0
- #define USB_CFG_DESCR_PROPS_UNKNOWN 0

5.4.1 Define Documentation

5.4.1.1 #define USB_CFG_DESCR_PROPS_CONFIGURATION 0

Definition at line 188 of file usbconfig.h.

5.4.1.2 #define USB_CFG_DESCR_PROPS_DEVICE 0

Definition at line 187 of file usbconfig.h.

5.4.1.3 #define USB_CFG_DESCR_PROPS_HID 0

Definition at line 194 of file usbconfig.h.

5.4.1.4 #define USB_CFG_DESCR_PROPS_HID_REPORT 0

Definition at line 195 of file usbconfig.h.

5.4.1.5 #define USB_CFG_DESCR_PROPS_STRING_0 0

Definition at line 190 of file usbconfig.h.

5.4.1.6 #define USB_CFG_DESCR_PROPS_STRING_PRODUCT 0

Definition at line 192 of file usbconfig.h.

5.4.1.7 #define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER 0

Definition at line 193 of file usbconfig.h.

5.4.1.8 #define USB_CFG_DESCR_PROPS_STRING_VENDOR 0

Definition at line 191 of file usbconfig.h.

5.4.1.9 #define USB_CFG_DESCR_PROPS_STRINGS 0

Definition at line 189 of file usbconfig.h.

5.4.1.10 #define USB_CFG_DESCR_PROPS_UNKNOWN 0

Definition at line 196 of file usbconfig.h.

5.4.1.11 #define USB_CFG_DEVICE_CLASS 0xff

Definition at line 128 of file usbconfig.h.

5.4.1.12 #define USB_CFG_DEVICE_ID 0xdc, 0x05

Definition at line 94 of file usbconfig.h.

5.4.1.13 #define USB_CFG_DEVICE_NAME 'U', 'S', 'B', 'a', 's', 'p'

Definition at line 114 of file usbconfig.h.

5.4.1.14 #define USB_CFG_DEVICE_NAME_LEN 6

Definition at line 115 of file usbconfig.h.

5.4.1.15 #define USB_CFG_DEVICE_SUBCLASS 0

Definition at line 129 of file usbconfig.h.

5.4.1.16 #define USB_CFG_DEVICE_VERSION 0x02, 0x01

Definition at line 101 of file usbconfig.h.

5.4.1.17 #define USB_CFG_HAVE_FLOWCONTROL 0

Definition at line 81 of file usbconfig.h.

5.4.1.18 #define USB_CFG_HAVE_INTRIN_ENDPOINT 0

Definition at line 36 of file usbconfig.h.

5.4.1.19 #define USB_CFG_HAVE_INTRIN_ENDPOINT3 0

Definition at line 40 of file usbconfig.h.

5.4.1.20 #define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH 0

Definition at line 138 of file usbconfig.h.

5.4.1.21 #define USB_CFG_IMPLEMENT_FN_READ 1

Definition at line 70 of file usbconfig.h.

5.4.1.22 #define USB_CFG_IMPLEMENT_FN_WRITE 1

Definition at line 65 of file usbconfig.h.

5.4.1.23 #define USB_CFG_IMPLEMENT_FN_WRITEOUT 0

Definition at line 76 of file usbconfig.h.

5.4.1.24 #define USB_CFG_IMPLEMENT_HALT 0

Definition at line 45 of file usbconfig.h.

5.4.1.25 #define USB_CFG_INTERFACE_CLASS 0

Definition at line 132 of file usbconfig.h.

5.4.1.26 #define USB_CFG_INTERFACE_PROTOCOL 0

Definition at line 134 of file usbconfig.h.

5.4.1.27 #define USB_CFG_INTERFACE_SUBCLASS 0

Definition at line 133 of file usbconfig.h.

5.4.1.28 #define USB_CFG_INTR_POLL_INTERVAL 200

Definition at line 51 of file usbconfig.h.

5.4.1.29 #define USB_CFG_IS_SELF_POWERED 0

Definition at line 56 of file usbconfig.h.

5.4.1.30 #define USB_CFG_MAX_BUS_POWER 100

Definition at line 60 of file usbconfig.h.

5.4.1.31 #define USB_CFG_VENDOR_ID 0xc0, 0x16

Definition at line 89 of file usbconfig.h.

5.4.1.32 #define USB_CFG_VENDOR_NAME 'w', 'w', 'w', '.', 'f', 'i', 's', 'c', 'h', 'l', '.', 'd', 'e'

Definition at line 104 of file usbconfig.h.

5.4.1.33 #define USB_CFG_VENDOR_NAME_LEN 13

Definition at line 105 of file usbconfig.h.

5.4.1.34 #define USB_PUBLIC static

Definition at line 29 of file usbconfig.h.

5.5 firmware/usbconfig.h File Reference

Configuration options for the USB-driver.

Defines

- #define USB_CFG_IOPORTNAME D

  This is the port where the USB bus is connected.

- #define USB_CFG_DMINUS_BIT 0

  This is the bit number in USB_CFG_IOPORT where the USB D- line is connected.

- #define USB_CFG_DPLUS_BIT 2

  This is the bit number in USB_CFG_IOPORT where the USB D+ line is connected.

- #define USB_CFG_HAVE_INTRIN_ENDPOINT 1

  Define this to 1 if you want to compile a version with two endpoints: The default control endpoint 0 and an interrupt-in endpoint 1.

- #define USB_CFG_HAVE_INTRIN_ENDPOINT3 0

  Define this to 1 if you want to compile a version with three endpoints: The default control endpoint 0, an interrupt-in endpoint 1 and an interrupt-in endpoint 3.

- #define USB_CFG_IMPLEMENT_HALT 0

  Define this to 1 if you also want to implement the ENDPOINT_HALT feature for endpoint 1 (interrupt endpoint).

- #define USB_CFG_INTR_POLL_INTERVAL 10

  If you compile a version with endpoint 1 (interrupt-in), this is the poll interval.

- #define USB_CFG_IS_SELF_POWERED 0

  Define this to 1 if the device has its own power supply.

- #define USB_CFG_MAX_BUS_POWER 100

  Set this variable to the maximum USB bus power consumption of your device.

- #define USB_CFG_IMPLEMENT_FN_WRITE 1

  Set this to 1 if you want usbFunctionWrite() to be called for control-out transfers.

- #define USB_CFG_IMPLEMENT_FN_READ 0

  Set this to 1 if you need to send control replies which are generated "on the fly" when usbFunctionRead() is called.

- #define USB_CFG_IMPLEMENT_FN_WRITEOUT 0

  Define this to 1 if you want to use interrupt-out (or bulk out) endpoint 1.

- #define USB_CFG_HAVE_FLOWCONTROL 0

  Define this to 1 if you want flowcontrol over USB data.

- #define USB_CFG_VENDOR_ID 0x42, 0x42

We cannot use Obdev's free shared VID/PID pair because this is a HID.

- #define USB_CFG_DEVICE_ID 0x31, 0xe1

  This is the ID of the product, low byte first.

- #define USB_CFG_DEVICE_VERSION 0x00, 0x01

  Version number of the device: Minor number first, then major number.

- #define USB_CFG_VENDOR_NAME 'w', 'w', 'w', '.', 's', 'c', 'h', 'a', 't', 'e', 'n', 's', 'e', 'i', 't', 'e', '.', 'd', 'e'

  These two values define the vendor name returned by the USB device.

- #define USB_CFG_VENDOR_NAME_LEN 19

  Length of USB_CFG_DEVICE_VERSION.

- #define USB_CFG_DEVICE_NAME 'D', 'u', 'l', 'c', 'i', 'm', 'e', 'r'

  Same as above for the device name.

- #define USB_CFG_DEVICE_NAME_LEN 8

  Length of USB_CFG_DEVICE_NAME.

- #define USB_CFG_DEVICE_CLASS 0

  See USB specification if you want to conform to an existing device class.

- #define USB_CFG_DEVICE_SUBCLASS 0

  See USB specification if you want to conform to an existing device subclass.

- #define USB_CFG_INTERFACE_CLASS 0x03

  See USB specification if you want to conform to an existing device class or protocol.

- #define USB_CFG_INTERFACE_SUBCLASS 0x01

  See USB specification if you want to conform to an existing device class or protocol.

- #define USB_CFG_INTERFACE_PROTOCOL 0x01

  See USB specification if you want to conform to an existing device class or protocol.

- #define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH 63

  Define this to the length of the HID report descriptor, if you implement an HID device.

- #define USB_CFG_DESCR_PROPS_DEVICE 0
- #define USB_CFG_DESCR_PROPS_CONFIGURATION 0
- #define USB_CFG_DESCR_PROPS_STRINGS 0
- #define USB_CFG_DESCR_PROPS_STRING_0 0
- #define USB_CFG_DESCR_PROPS_STRING_VENDOR 0
- #define USB_CFG_DESCR_PROPS_STRING_PRODUCT 0
- #define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER 0
- #define USB_CFG_DESCR_PROPS_HID 0
- #define USB_CFG_DESCR_PROPS_HID_REPORT 0
- #define USB_CFG_DESCR_PROPS_UNKNOWN 0

5.5.1 Detailed Description

Configuration options for the USB-driver.

This file is almost identical to the original usbconfig.h by Christian Starkjohann, in structure and content.

It contains parts of the USB driver which can be configured and can or must be adapted to your hardware.

Author:

Ronald Schaten <ronald@schatenseite.de>

Version:

Id

usbconfig.h,v 1.1 2008-07-09 20:47:12 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file usbconfig.h.

5.5.2 Define Documentation

5.5.2.1 #define USB_CFG_DESCR_PROPS_CONFIGURATION 0

Definition at line 214 of file usbconfig.h.

5.5.2.2 #define USB_CFG_DESCR_PROPS_DEVICE 0

Definition at line 213 of file usbconfig.h.

5.5.2.3 #define USB_CFG_DESCR_PROPS_HID 0

Definition at line 220 of file usbconfig.h.

5.5.2.4 #define USB_CFG_DESCR_PROPS_HID_REPORT 0

Definition at line 221 of file usbconfig.h.

5.5.2.5 #define USB_CFG_DESCR_PROPS_STRING_0 0

Definition at line 216 of file usbconfig.h.

5.5.2.6 #define USB_CFG_DESCR_PROPS_STRING_PRODUCT 0

Definition at line 218 of file usbconfig.h.

5.5.2.7 #define USB_CFG_DESCR_PROPS_STRING_SERIAL_NUMBER 0

Definition at line 219 of file usbconfig.h.

5.5.2.8 #define USB_CFG_DESCR_PROPS_STRING_VENDOR 0

Definition at line 217 of file usbconfig.h.

5.5.2.9 #define USB_CFG_DESCR_PROPS_STRINGS 0

Definition at line 215 of file usbconfig.h.

5.5.2.10 #define USB_CFG_DESCR_PROPS_UNKNOWN 0

Definition at line 222 of file usbconfig.h.

5.5.2.11 #define USB_CFG_DEVICE_CLASS 0

See USB specification if you want to conform to an existing device class.

This setting means to specify the class at the interface level.

Definition at line 148 of file usbconfig.h.

5.5.2.12 #define USB_CFG_DEVICE_ID 0x31, 0xe1

This is the ID of the product, low byte first.

It is interpreted in the scope of the vendor ID. If you have registered your own VID with usb.org or if you have licensed a PID from somebody else, define it here. Otherwise you use obdev's free shared VID/PID pair. Be sure to read the rules in USBID-License.txt!

Definition at line 122 of file usbconfig.h.

5.5.2.13 #define USB_CFG_DEVICE_NAME 'D', 'u', 'l', 'c', 'i', 'm', 'e', 'r'

Same as above for the device name.

If you don't want a device name, undefine the macros. See the file USBID-License.txt before you assign a name.

Definition at line 141 of file usbconfig.h.

5.5.2.14 #define USB_CFG_DEVICE_NAME_LEN 8

Length of USB_CFG_DEVICE_NAME.

Definition at line 144 of file usbconfig.h.

5.5.2.15 #define USB_CFG_DEVICE_SUBCLASS 0

See USB specification if you want to conform to an existing device subclass.

Definition at line 151 of file usbconfig.h.

5.5.2.16 #define USB_CFG_DEVICE_VERSION 0x00, 0x01

Version number of the device: Minor number first, then major number.

Definition at line 125 of file usbconfig.h.

5.5.2.17 #define USB_CFG_DMINUS_BIT 0

This is the bit number in USB_CFG_IOPORT where the USB D- line is connected.

This may be any bit in the port.

Definition at line 28 of file usbconfig.h.

5.5.2.18 #define USB_CFG_DPLUS_BIT 2

This is the bit number in USB_CFG_IOPORT where the USB D+ line is connected.

This may be any bit in the port. Please note that D+ must also be connected to interrupt pin INT0!

Definition at line 33 of file usbconfig.h.

5.5.2.19 #define USB_CFG_HAVE_FLOWCONTROL 0

Define this to 1 if you want flowcontrol over USB data.

See the definition of the macros usbDisableAllRequests() and usbEnableAllRequests() in usbdrv.h.

Definition at line 100 of file usbconfig.h.

5.5.2.20 #define USB_CFG_HAVE_INTRIN_ENDPOINT 1

Define this to 1 if you want to compile a version with two endpoints: The default control endpoint 0 and an interrupt-in endpoint 1.

Definition at line 54 of file usbconfig.h.

5.5.2.21 #define USB_CFG_HAVE_INTRIN_ENDPOINT3 0

Define this to 1 if you want to compile a version with three endpoints: The default control endpoint 0, an interrupt-in endpoint 1 and an interrupt-in endpoint 3.

You must also enable endpoint 1 above.

Definition at line 59 of file usbconfig.h.

5.5.2.22 #define USB_CFG_HID_REPORT_DESCRIPTOR_LENGTH 63

Define this to the length of the HID report descriptor, if you implement an HID device.

Otherwise don't define it or define it to 0.

Definition at line 167 of file usbconfig.h.

5.5.2.23 #define USB_CFG_IMPLEMENT_FN_READ 0

Set this to 1 if you need to send control replies which are generated "on the fly" when usbFunctionRead() is called.

If you only want to send data from a static buffer, set it to 0 and return the data from usbFunctionSetup(). This saves a couple of bytes.

Definition at line 90 of file usbconfig.h.

5.5.2.24 #define USB_CFG_IMPLEMENT_FN_WRITE 1

Set this to 1 if you want usbFunctionWrite() to be called for control-out transfers.

Set it to 0 if you don't need it and want to save a couple of bytes.

Definition at line 84 of file usbconfig.h.

5.5.2.25 #define USB_CFG_IMPLEMENT_FN_WRITEOUT 0

Define this to 1 if you want to use interrupt-out (or bulk out) endpoint 1.

You must implement the function usbFunctionWriteOut() which receives all interrupt/bulk data sent to endpoint 1.

Definition at line 95 of file usbconfig.h.

5.5.2.26 #define USB_CFG_IMPLEMENT_HALT 0

Define this to 1 if you also want to implement the ENDPOINT_HALT feature for endpoint 1 (interrupt endpoint).

Although you may not need this feature, it is required by the standard. We have made it a config option because it bloats the code considerably.

Definition at line 65 of file usbconfig.h.

5.5.2.27 #define USB_CFG_INTERFACE_CLASS 0x03

See USB specification if you want to conform to an existing device class or protocol.

This is HID class.

Definition at line 155 of file usbconfig.h.

5.5.2.28 #define USB_CFG_INTERFACE_PROTOCOL 0x01

See USB specification if you want to conform to an existing device class or protocol.

This is keyboard protocol.

Definition at line 163 of file usbconfig.h.

5.5.2.29 #define USB_CFG_INTERFACE_SUBCLASS 0x01

See USB specification if you want to conform to an existing device class or protocol.

This is a boot device.

Definition at line 159 of file usbconfig.h.

5.5.2.30 #define USB_CFG_INTR_POLL_INTERVAL 10

If you compile a version with endpoint 1 (interrupt-in), this is the poll interval.

The value is in milliseconds and must not be less than 10 ms for low speed devices.

Definition at line 70 of file usbconfig.h.

5.5.2.31 #define USB_CFG_IOPORTNAME D

This is the port where the USB bus is connected.

When you configure it to "B", the registers PORTB, PINB and DDRB will be used.

Definition at line 24 of file usbconfig.h.

5.5.2.32 #define USB_CFG_IS_SELF_POWERED 0

Define this to 1 if the device has its own power supply.

Set it to 0 if the device is powered from the USB bus.

Definition at line 74 of file usbconfig.h.

5.5.2.33 #define USB_CFG_MAX_BUS_POWER 100

Set this variable to the maximum USB bus power consumption of your device.

The value is in milliamperes. [It will be divided by two since USB communicates power requirements in units of 2 mA.]

Definition at line 79 of file usbconfig.h.

5.5.2.34 #define USB_CFG_VENDOR_ID 0x42, 0x42

We cannot use Obdev's free shared VID/PID pair because this is a HID.

We use John Hyde's VID (author of the book "USB Design By Example") for this example instead. John has offered this VID for use by students for non-commercial devices. Well... This example is for demonstration and education only... DO NOT LET DEVICES WITH THIS VID ESCAPE YOUR LAB! The Product-ID is a random number.

USB vendor ID for the device, low byte first. If you have registered your own Vendor ID, define it here. Otherwise you use obdev's free shared VID/PID pair. Be sure to read USBID-License.txt for rules!

Definition at line 115 of file usbconfig.h.

5.5.2.35 #define USB_CFG_VENDOR_NAME 'w', 'w', 'w', '.', 's', 'c', 'h', 'a', 't', 'e', 'n', 's', 'e', 'i', 't', 'e', '.', 'd', 'e'

These two values define the vendor name returned by the USB device.

The name must be given as a list of characters under single quotes. The characters are interpreted as Unicode (UTF-16) entities. If you don't want a vendor name string, undefine these macros. ALWAYS define a vendor name containing your Internet domain name if you use obdev's free shared VID/PID pair. See the file USBID-License.txt for details.

Definition at line 134 of file usbconfig.h.

5.5.2.36 #define USB_CFG_VENDOR_NAME_LEN 19

Length of USB_CFG_DEVICE_VERSION.

Definition at line 137 of file usbconfig.h.

5.6 firmware/keycodes.h File Reference

This file contains modifier- and keycode definitions according to the USB-specifications for human interface devices.

Enumerations

- enum modifiers {

  MOD_NONE = 0, MOD_CONTROL_LEFT = (1 << 0), MOD_SHIFT_LEFT = (1 << 1), MOD_ALT_LEFT = (1 << 2),

  MOD_GUI_LEFT = (1 << 3), MOD_CONTROL_RIGHT = (1 << 4), MOD_SHIFT_RIGHT = (1 << 5), MOD_ALT_RIGHT = (1 << 6),

  MOD_GUI_RIGHT = (1 << 7) }

  Codes for modifier-keys.

- enum keycodes {

  KEY_Reserved = 0, KEY_ErrorRollOver, KEY_POSTFail, KEY_ErrorUndefined,

  KEY_A, KEY_B, KEY_C, KEY_D,

  KEY_E, KEY_F, KEY_G, KEY_H,

  KEY_I, KEY_J, KEY_K, KEY_L,

  KEY_M, KEY_N, KEY_O, KEY_P,

  KEY_Q, KEY_R, KEY_S, KEY_T,

  KEY_U, KEY_V, KEY_W, KEY_X,

  KEY_Y, KEY_Z, KEY_1, KEY_2,

  KEY_3, KEY_4, KEY_5, KEY_6,

  KEY_7, KEY_8, KEY_9, KEY_0,

  KEY_Return, KEY_ESCAPE, KEY_DELETE, KEY_Tab,

  KEY_Spacebar, KEY_minus, KEY_equals, KEY_lbracket,

  KEY_rbracket, KEY_backslash, KEY_hash, KEY_semicolon,

  KEY_apostroph, KEY_grave, KEY_comma, KEY_dot,

  KEY_slash, KEY_capslock, KEY_F1, KEY_F2,

  KEY_F3, KEY_F4, KEY_F5, KEY_F6,

  KEY_F7, KEY_F8, KEY_F9, KEY_F10,

  KEY_F11, KEY_F12, KEY_PrintScreen, KEY_ScrollLock,

  KEY_Pause, KEY_Insert, KEY_Home, KEY_PageUp,

  KEY_DeleteForward, KEY_End, KEY_PageDown, KEY_RightArrow,

  KEY_LeftArrow, KEY_DownArrow, KEY_UpArrow, KEY_NumLock,

  KEY_KPslash, KEY_KPasterisk, KEY_KPminus, KEY_KPplus,

  KEY_KPenter, KEY_KP1, KEY_KP2, KEY_KP3,

  KEY_KP4, KEY_KP5, KEY_KP6, KEY_KP7,

  KEY_KP8, KEY_KP9, KEY_KP0, KEY_KPcomma,

  KEY_Euro, KEY_Application }

  Codes for non-modifier-keys.

5.6.1 Detailed Description

This file contains modifier- and keycode definitions according to the USB-specifications for human interface devices.

See usb.org's HID-usage-tables document, chapter 10 Keyboard/Keypad Page for more codes: http://www.usb.org/developers/devclass_docs/Hut1_12.pdf

Author:

Ronald Schaten <ronald@schatenseite.de>

Version:

Id

keycodes.h,v 1.1 2008-07-09 20:47:12 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file keycodes.h.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum keycodes

Codes for non-modifier-keys.

Enumerator:

KEY_Reserved

KEY_ErrorRollOver

KEY_POSTFail

KEY_ErrorUndefined

KEY_A

KEY_B

KEY_C

KEY_D

KEY_E

KEY_F

KEY_G

KEY_H

KEY_I

KEY_J

KEY_K

KEY_L

KEY_M

KEY_N

KEY_O
KEY_P
KEY_Q
KEY_R
KEY_S
KEY_T
KEY_U
KEY_V
KEY_W
KEY_X
KEY_Y
KEY_Z
KEY_1
KEY_2
KEY_3
KEY_4
KEY_5
KEY_6
KEY_7
KEY_8
KEY_9
KEY_0
KEY_Return
KEY_ESCAPE
KEY_DELETE
KEY_Tab
KEY_Spacebar
KEY_minus
KEY_equals
KEY_lbracket
KEY_rbracket
KEY_backslash
KEY_hash
KEY_semicolon
KEY_apostroph
KEY_grave
KEY_comma
KEY_dot
KEY_slash
KEY_capslock
KEY_F1

KEY_F2
KEY_F3
KEY_F4
KEY_F5
KEY_F6
KEY_F7
KEY_F8
KEY_F9
KEY_F10
KEY_F11
KEY_F12
KEY_PrintScreen
KEY_ScrollLock
KEY_Pause
KEY_Insert
KEY_Home
KEY_PageUp
KEY_DeleteForward
KEY_End
KEY_PageDown
KEY_RightArrow
KEY_LeftArrow
KEY_DownArrow
KEY_UpArrow
KEY_NumLock
KEY_KPslash
KEY_KPasterisk
KEY_KPminus
KEY_KPplus
KEY_KPenter
KEY_KP1
KEY_KP2
KEY_KP3
KEY_KP4
KEY_KP5
KEY_KP6
KEY_KP7
KEY_KP8
KEY_KP9
KEY_KP0
KEY_KPcomma
KEY_Euro
KEY_Application

Definition at line 31 of file keycodes.h.

5.6.2.2 enum modifiers

Codes for modifier-keys.

Enumerator:

MOD_NONE
MOD_CONTROL_LEFT
MOD_SHIFT_LEFT
MOD_ALT_LEFT
MOD_GUI_LEFT
MOD_CONTROL_RIGHT
MOD_SHIFT_RIGHT
MOD_ALT_RIGHT
MOD_GUI_RIGHT

Definition at line 18 of file keycodes.h.

Index