PERswitch

0.8

Generated by Doxygen 1.7.6.1

Fri Jun 8 2012 15:37:57

Contents

Chapter 1

PERswitch

Date

31 Oct. 2010

Author

C. Bischko

Note

This is not the final version of the project. Inspect everything carefully before you use it. Especially make sure, that the boards are right (fit together and are not mirror-inverted on one side)! The documentation on the windows client and driver is missing. Consult the obdev website to fill the gaps. I lost dirve to do that properly since I'm not using windows anymore and the prototype worked.

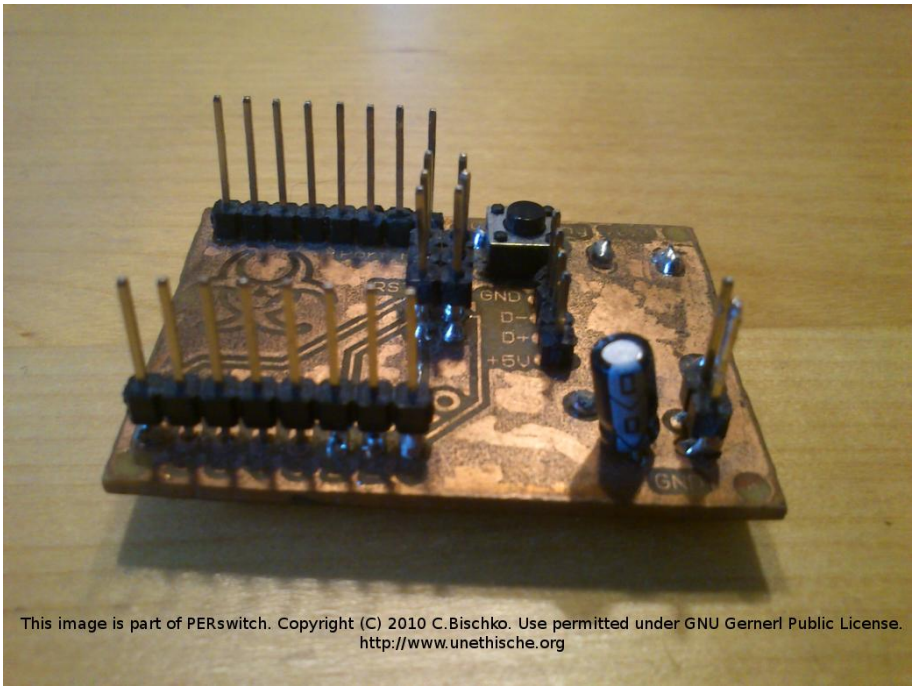


Figure 1.1: Front of the Controller board

1.1 Introduction

I grew tired of crouching under my desk to power my printer, scanner, sound and external HDDs. So with much inspiration form other projects I came up with this one.

The periphery switch (PERswitch for short) is a project consisting of two boards: a relay card and a controller board. Its intended use is the switching of 230 V AC / 50 Hz (-European Standard) periphery from a PC via USB. Since all periphery will be powered by the switch it might be a good idea to include a cheap usb hub. This way the usb cables plug into the switch as well and you kill two birds with one cable.

PERswitch is fairly easy to build, even for beginners and a great way to get into electronics, since you end up with a cheap product, that's usefull to almost anybody. Building it will cost you about 10,- € and a day of your time. It uses both surface mount and DIP components. So you will need the propper tools to handle them as well as the means to etch double sided PCBs at home.

Note

Most of this project is based on other projects. Most prominently the examples that come with VUSB by Objetive Developement. Thaks a lot guys!

Warning

This projects works with 230 V AC at 50 Hz. This is potentially dangerous. Handle with care and know, what you're doing.

1.2 Manufacturing the Boards

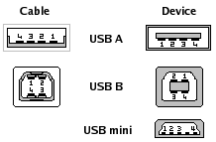
I will not go into much detail here. Form the links below you can learn the necessary skills to build the boards yourself:

- Make single sided PCBs
- Make double sided PCBs (In German - but you'll get the drift of things)
- Learn how to solder

For beginners I recommend to start with the relay board. It is single sided and therefore easier to etch. Also it uses through hole components only, which makes it a good soldering exercise for beginners.

You will find the layouts in the *'board/'* directory. The scematics specify the components you'll need and where they go. Eagle files are included in *'board/eagle/'* in case you might want to modify the design. Eagle is free for private users.

I included a cheap noname 5 port usb-hub into my own PERswitch. It is fairly easy to identify the usb-lines on one of those (with the help of the figure below). Just desolder one of the connectors. Then you can wire the hub directly to to the controller board. You can use the wall plug that comes with your hub to power all the boards. You might want to desolder all the hub's connectors and solder cables directly to the boards in order to conserve space.



Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

Figure 1.2: Identify the usb lines

1.3 Compiling and flashing the firmware

The firmware for this project requires 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.

Lady Ada's avr tutorial covers the basics of flashing firmware to a micro-controller nicely. Note that the controller board comes with a 6-pin ISP-Header so you can connect your programmer directly to the board.

However, running "make program" in the *firmware* direcotry should be sufficient. Maybe you will have to edit the Makefile especially if you are using a programmer other than USBtinyISP (btw.: you can build USBtinyISP at home).

1.4 Clientsoftware

Source code and binaries are included for both Ubuntu 10.10 and Windows (32 bit versions). The clientsoftware consists of a command line tool and a graphical user interface (GUI). The GUI requires the gtk2 toolkit both executables require libusb.

Here is a screen shot of the status icon created the GUI. Each entry in the pull down menu corresponds to one of the relays. It will toggle the respective device on and off. There is a configuration file to edit the labels (or you might want to edit the source code),

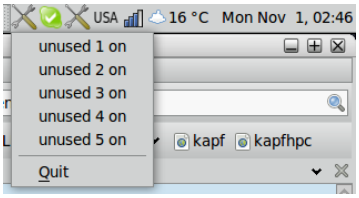


Figure 1.3: Screenshot of the client program

1.5 Hooking it up

Warning

To use the board, you need to hook it up to the power line at some point. Please be extremely carefull. Be sure to read and understand the entirety of this section before you work with 230V. Use your common sense. Wall plug electricity can kill you! (and is extremely unpleasant even if it doesn't).

The first thing you will want to do is to test the boards. Using only the 5 V supplied by the usb connector on your computer is fine. So no wall plug 230 V at first! 5 Volts are enough to drive the relays and you will hear a satisfying click sound from them, if you assembeld and hooked up everything right.

Now on the controller board the pins labeled 0 to 4 of "Port A" connect to the 5 pin header labeld "1,2,3,4,5" on the relay board. The Pins labeld "+5V" and "GND" on the contoller board have to be connected to their counterparts on the relay board. If everything went fine you sould be able to switch the channels using *'clientsoftware/PERScom'* (or 'PERScom.exe' on Windows).

Once this test is done, you can connect your periphery devices and the wall plug cord to the screw terminals. Do not plug it into the wall just now and use the proper eylets (kabel shoes) for your bare wires. V0 is the label for wall plug ground (yellow/green wire), V+ is the AC conductor (brown insulation) and V- the zero conducor (blue wire). The latter two are basically interchangable for AC.

Now you have everything hooked up. You neew to put it in a non conducting, robust case, make sure that conducors on 230V AC potential never touch other conducors (and will never do so in the future). **Make sure that no one - keep children in mind - will be able to touch the 230V ciruitry by accident.**

1.6 Stuff

I don't guaratee that any of this will work for anyone. Neither do I feel responsible for any damage caused by the use or missuse of this project or the software provided with it. You are free to use and modify the project in any way you deem appropriate, provided the licenses metioned in the respective files don't say otherwise.

Take care and have fun.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 clientsoftware/cmdline.c File Reference

Source file for cmdline.h.

```
#include <stdio.h> #include <stdlib.h> #include <string.-
h> #include "cmdline.h" #include "opendevice.h" #include
"../firmware/requests.h" #include "../firmware/usbconfig.-
h"
```

Functions

- void usage (char *name)

  Prints how to use the command line client.
- int createHandle (usb_dev_handle **handle)

  Creates the device handle for PERswitch.
- int doRequest (usb_dev_handle *handle, unsigned int portNbr, unsigned int pinNbr, char *command, int *status)

  Does the actual request.

3.1.1 Detailed Description

Source file for cmdline.h. Most of this is based on Objective Development's VUSB software and examples.

Date

31. Oct. 2010

Author

C. Bischko

Copyright

3.1.2 Function Documentation

3.1.2.1 int createHandle (usb_dev_handle ** *handle*)

Creates the device handle for PERswitch.

Parameters

handle	Pointer to the future handle

Returns

0 if the handle was created, 1 otherwise

3.1.2.2 int doRequest (usb_dev_handle * *handle,* unsigned int *portNbr,* unsigned int *pinNbr,* char * *command,* int * *status*)

Does the actual request.

Switches a pin #pinNbr at port #portNbr on, off or requests its status.

Parameters

handle	Handle created by

See also

createHandle

Parameters

portNbr	Number of the port the request is about
pinNbr	Number of the pin the request is about
command	Action that is to be performed (either "on", "off" or "status")

Returns

0 if the request was successfull, 1 otherwise

3.1.2.3 void usage (char * *name*)

Prints how to use the command line client.

3.2 clientsoftware/cmdline.h File Reference

PERswitch utility functions.

`#include <usb.h>`

Functions

- void usage (char *name)

  Prints how to use the command line client.
- int createHandle (usb_dev_handle **handle)

  Creates the device handle for PERswitch.
- int doRequest (usb_dev_handle *handle, unsigned int portNbr, unsigned int pinNbr, char *command, int *status)

  Does the actual request.

3.2.1 Detailed Description

PERswitch utility functions. Used by both the PERcom command line client and the PERgui tray program.

Most of this is based on Objective Development's VUSB software and examples.

Date

31. Oct. 2010

Author

C. Bischko

Copyright

This file is part of PERswitch. Copyright © 2012 C. Bischko. Use permitted under GNU General Public License.

3.2.2 Function Documentation

3.2.2.1 int createHandle (usb_dev_handle ** *handle*)

Creates the device handle for PERswitch.

Parameters

handle	Pointer to the future handle

Returns

0 if the handle was created, 1 otherwise

3.2.2.2 int doRequest (usb_dev_handle * *handle*, unsigned int *portNbr*, unsigned int *pinNbr*, char * *command*, int * *status*)

Does the actual request.

Switches a pin #pinNbr at port #portNbr on, off or requests its status.

Parameters

handle	Handle created by

See also

createHandle

Parameters

portNbr	Number of the port the request is about
pinNbr	Number of the pin the request is about
command	Action that is to be performed (either "on", "off" or "status")

Returns

0 if the request was successfull, 1 otherwise

3.2.2.3 void usage (char * *name*)

Prints how to use the command line client.

3.3 clientsoftware/opendevice.c File Reference

Source file to the header file opendevice.h.

```
#include <stdio.h> #include "opendevice.h"
```

Defines

- #define MATCH_SUCCESS 1

  Internal return code for a found device.
- #define MATCH_FAILED 0

  Internal error code for device not found.

- #define MATCH_ABORT -1

  Internal error code for aborted device search.

Functions

- static int _shellStyleMatch (char *text, char *p)

  Private interface: match text and p.
- static int shellStyleMatch (char *text, char *pattern)

  Public interface for shell style matching.
- int usbGetStringAscii (usb_dev_handle *dev, int index, char *buf, int buflen)

  This function gets a string descriptor from the device.
- int usbOpenDevice (usb_dev_handle **device, int vendorID, char *vendorNamePattern, int productID, char *productNamePattern, char *serialNamePattern, FILE *printMatchingDevicesFp, FILE *warningsFp)

  This function iterates over all devices on all USB busses and searches for a device.

3.3.1 Detailed Description

Source file to the header file opendevice.h. Copyright: (c) 2008 by OBJECTIVE DEVELOPMENT Software GmbH

License: GNU GPL v2 (see License.txt), GNU GPL v3 or proprietary (CommercialLicense.txt)

Date

2008-04-10

Author

Christian Starkjohann

Version

Id:

opendevice.h 755 2009-08-03 17:01:21Z cs

3.3.2 Define Documentation

3.3.2.1 #define MATCH_ABORT -1

Internal error code for aborted device search.

3.3.2.2 #define MATCH_FAILED 0

Internal error code for device not found.

3.3.2.3 #define MATCH_SUCCESS 1

Internal return code for a found device.

3.3.3 Function Documentation

3.3.3.1 static int _shellStyleMatch (char * *text,* char * *p*) [static]

Private interface: match text and p.

Returns

MATCH_SUCCESS, MATCH_FAILED, or MATCH_ABORT.

3.3.3.2 static int shellStyleMatch (char * *text,* char * *pattern*) [static]

Public interface for shell style matching.

Returns

0 if fails, 1 if matches

3.3.3.3 int usbGetStringAscii (usb_dev_handle * *dev,* int *index,* char * *buf,* int *buflen*)

This function gets a string descriptor from the device.

Parameters

dev	libusb device handle for the device
index	The string descriptor index.
buf	Contains the string in ISO Latin 1 encoding terminated with a 0-character.
buflen	length of the buffer

Returns

length of the string (excluding the terminating 0) or a negative number in case of an error. If there was an error, use usb_strerror() to obtain the error message.

3.3.3.4 int usbOpenDevice (usb_dev_handle ** *device*, int *vendorID*, char * *vendorNamePattern*, int *productID*, char * *productNamePattern*, char * *serialNamePattern*, FILE * *printMatchingDevicesFp*, FILE * *warningsFp*)

This function iterates over all devices on all USB busses and searches for a device.

Matching is done first by means of Vendor- and Product-ID (passed in 'vendorID' and 'productID'. An ID of 0 matches any numeric ID (wildcard). When a device matches by its IDs, matching by names is performed. Name matching can be done on textual vendor name ('vendorNamePattern'), product name ('productNamePattern') and serial number ('serialNamePattern'). A device matches only if all non-null pattern match. - If you don't care about a string, pass NULL for the pattern. Patterns are Unix shell style pattern: '*' stands for 0 or more characters, '?' for one single character, a list of characters in square brackets for a single character from the list (dashes are allowed to specify a range) and if the lis of characters begins with a caret ('^'), it matches one character which is NOT in the list. Other parameters to the function: If 'warningsFp' is not NULL, warning messages are printed to this file descriptor with fprintf(). If 'printMatchingDevicesFp' is not NULL, no device is opened but matching devices are printed to the given file descriptor with fprintf(). If a device is opened, the resulting USB handle is stored in '*device'. A pointer to a "usb_dev_handle *" type variable must be passed here.

Returns

0 on success, an error code (see defines) on failure.

3.4 clientsoftware/opendevice.h File Reference

V-USB host-side library.

`#include <usb.h> #include <stdio.h>`

Defines

- #define USBOPEN_SUCCESS 0

  no error
- #define USBOPEN_ERR_ACCESS 1

  not enough permissions to open device
- #define USBOPEN_ERR_IO 2

  I/O error.
- #define USBOPEN_ERR_NOTFOUND 3

  device not found
- #define USB_VID_OBDEV_SHARED 5824

  obdev's shared vendor ID
- #define USB_PID_OBDEV_SHARED_CUSTOM 1500

  shared PID for custom class devices
- #define USB_PID_OBDEV_SHARED_HID 1503

shared PID for HIDs except mice & keyboards

- #define USB_PID_OBDEV_SHARED_CDCACM 1505

  shared PID for CDC Modem devices

- #define USB_PID_OBDEV_SHARED_MIDI 1508

  shared PID for MIDI class devices

Functions

- int usbGetStringAscii (usb_dev_handle *dev, int index, char *buf, int buflen)

  This function gets a string descriptor from the device.

- int usbOpenDevice (usb_dev_handle **device, int vendorID, char *vendorNamePattern, int productID, char *productNamePattern, char *serialNamePattern, FILE *printMatchingDevicesFp, FILE *warningsFp)

  This function iterates over all devices on all USB busses and searches for a device.

3.4.1 Detailed Description

V-USB host-side library. This module offers additional functionality for host side drivers based on libusb or libusb-win32. It includes a function to find and open a device based on numeric IDs and textual description. It also includes a function to obtain textual descriptions from a device.

To use this functionality, simply copy opendevice.c and opendevice.h into your project and add them to your Makefile. You may modify and redistribute these files according to the GNU General Public License (GPL) version 2 or 3.

Copyright: (c) 2008 by OBJECTIVE DEVELOPMENT Software GmbH

License: GNU GPL v2 (see License.txt), GNU GPL v3 or proprietary (CommercialLicense.txt)

Date

2008-04-10

Author

Christian Starkjohann

Version

Id:

opendevice.h 755 2009-08-03 17:01:21Z cs

3.4.2 Define Documentation

3.4.2.1 #define USB_PID_OBDEV_SHARED_CDCACM 1505

shared PID for CDC Modem devices

3.4.2.2 #define USB_PID_OBDEV_SHARED_CUSTOM 1500

shared PID for custom class devices

3.4.2.3 #define USB_PID_OBDEV_SHARED_HID 1503

shared PID for HIDs except mice & keyboards

3.4.2.4 #define USB_PID_OBDEV_SHARED_MIDI 1508

shared PID for MIDI class devices

3.4.2.5 #define USB_VID_OBDEV_SHARED 5824

obdev's shared vendor ID

3.4.2.6 #define USBOPEN_ERR_ACCESS 1

not enough permissions to open device

3.4.2.7 #define USBOPEN_ERR_IO 2

I/O error.

3.4.2.8 #define USBOPEN_ERR_NOTFOUND 3

device not found

3.4.2.9 #define USBOPEN_SUCCESS 0

no error

3.4.3 Function Documentation

3.4.3.1 int usbGetStringAscii (usb_dev_handle * *dev,* int *index,* char * *buf,* int *buflen*)

This function gets a string descriptor from the device.

Parameters

dev	libusb device handle for the device
index	The string descriptor index.
buf	Contains the string in ISO Latin 1 encoding terminated with a 0-character.
buflen	length of the buffer

Returns

length of the string (excluding the terminating 0) or a negative number in case of an error. If there was an error, use usb_strerror() to obtain the error message.

3.4.3.2 int usbOpenDevice (usb_dev_handle ** *device,* int *vendorID,* char * *vendorNamePattern,* int *productID,* char * *productNamePattern,* char * *serialNamePattern,* FILE * *printMatchingDevicesFp,* FILE * *warningsFp*)

This function iterates over all devices on all USB busses and searches for a device.

Matching is done first by means of Vendor- and Product-ID (passed in 'vendorID' and 'productID'. An ID of 0 matches any numeric ID (wildcard). When a device matches by its IDs, matching by names is performed. Name matching can be done on textual vendor name ('vendorNamePattern'), product name ('productNamePattern') and serial number ('serialNamePattern'). A device matches only if all non-null pattern match. - If you don't care about a string, pass NULL for the pattern. Patterns are Unix shell style pattern: '*' stands for 0 or more characters, '?' for one single character, a list of characters in square brackets for a single character from the list (dashes are allowed to specify a range) and if the lis of characters begins with a caret ('^'), it matches one character which is NOT in the list. Other parameters to the function: If 'warningsFp' is not NULL, warning messages are printed to this file descriptor with fprintf(). If 'printMatchingDevicesFp' is not NULL, no device is opened but matching devices are printed to the given file descriptor with fprintf(). If a device is opened, the resulting USB handle is stored in '*device'. A pointer to a "usb_dev_handle *" type variable must be passed here.

Returns

0 on success, an error code (see defines) on failure.

3.5 clientsoftware/PERgui.c File Reference

Controls the PERswitch periphery switch via a tray menu.

```
#include <stdlib.h> #include <stdio.h> #include <string.-
h> #include <gtk/gtk.h> #include <usb.h> #include "cmdline.-
h"
```

Defines

- #define MENU_SIZE 5
- #define PORT_NBR 2

Functions

- int readNames (void)
- void popNotFoundDialog (void)
- static void opemnenu (GtkStatusIcon *status_icon, guint button, guint activate_time, gpointer user_data)
- static void channelCallback (GtkStatusIcon *widget, gpointer data)
- int main (int argc, char **argv)

Variables

- char * itemNames [MENU_SIZE]
- int * pinStatus
- char * tmpItemNames [MENU_SIZE]
- GtkWidget ** menuItems

3.5.1 Detailed Description

Controls the PERswitch periphery switch via a tray menu. Creates a system tray icon. When clicked the icon pops a menu. Each menu item corresponds to one of the PERswitch's channels and switches them on and off.

On popup the menu refreshes the status of each channel automatically ("on" or "off").

Date

31. Oct. 2010

Author

C. Bischko

Copyright

3.5.2 Define Documentation

3.5.2.1 #define MENU_SIZE 5

3.5.2.2 #define PORT_NBR 2

3.5.3 Function Documentation

3.5.3.1 static void channelCallback (GtkStatusIcon * *widget,* gpointer *data*) [static]

3.5.3.2 int main (int *argc,* char ** *argv*)

3.5.3.3 static void opemnenu (GtkStatusIcon * *status_icon,* guint *button,* guint *activate_time,* gpointer *user_data*) [static]

3.5.3.4 void popNotFoundDialog (void)

3.5.3.5 int readNames (void)

3.5.4 Variable Documentation

3.5.4.1 char* itemNames[MENU_SIZE]

Initial value:

```
 {
    "Unnamed 1","Unnamed 2","Unnamed 3","Unnamed 4","Unnamed 5"
}
```

3.5.4.2 GtkWidget** menuItems

3.5.4.3 int* pinStatus

3.5.4.4 char* tmpItemNames[MENU_SIZE]

3.6 clientsoftware/PERScom.c File Reference

Command line client for PERswitch.

```
#include "cmdline.h"
```

Functions

- int main (int argc, char **argv)

  Main function of the command line client.

3.6.1 Detailed Description

Command line client for PERswitch. Used to communicate with the periphery switch over usb. Can be used to turn certain pins on a given port on/off or query their status.

Most of this is based on Objective Development's VUSB software and examples.

Date

31. Oct. 2010

Author

C. Bischko

Copyright

3.6.2 Function Documentation

3.6.2.1 int main (int *argc*, char ** *argv*)

Main function of the command line client.

3.7 firmware/main.c File Reference

MCU Firmware for PERswitch.

```
#include <avr/io.h> #include <avr/interrupt.h> #include
<util/delay.h> #include <avr/pgmspace.h> #include "osccal.-
h" #include "./usbdrv/usbdrv.h" #include "requests.h"
```

Defines

- #define HELP_CONCAT_MACROS(x, y) x##y

  Helper necessary to concat two macros.
- #define CONCAT_MACROS(x, y) HELP_CONCAT_MACROS(x,y)

  Fuses two Macros together.

Functions

- void writePin (volatile uint8_t *port, uint8_t pinNbr, uint8_t pinStat)

Pin-Setting-Function.

- uint8_t usbFunctionSetup (uint8_t data[8])

  USB-Setup-Handler.

- int main (void)

  Main-function.

3.7.1 Detailed Description

MCU Firmware for PERswitch. This contains the main magic on the divice's side. There are two usb requests implemented: CMD_GET and CMD_SET. Respectively they get or set the status on a given pin.

Note

There is a lot crammed into the tiny 2k of the ATtiny26 controller. Even minor changes in this code might cause the compiled version not to fit into the ATtiny's program memory.

Most of this is based on Objective Development's VUSB software and examples.

Date

31. Oct. 2010

Author

C. Bischko

Copyright

3.7.2 Define Documentation

3.7.2.1 #define CONCAT_MACROS(*x*, *y*) HELP_CONCAT_MACROS(x,y)

Fuses two Macros together.

3.7.2.2 #define HELP_CONCAT_MACROS(*x*, *y*) x##y

Helper necessary to concat two macros.

3.7.3 Function Documentation

3.7.3.1 int main (void)

Main-function.

Initializes the hardware and starts the main loop of the application. Nothing terribly exciting here.

Returns

Error code - nothing in this case.

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

USB-Setup-Handler.

Handles setup-calls that are received from the USB-Interface i.e. whole communication between the device and the host.

Parameters

data	Eight bytes of data: (byte, name, descriton) • 0 = bmRequestType, Bitmask containing information about the context. • 1 = bRequest, User specified number to identify request. • 2-5 = wValue wIndex, In vendor requests this can be arbitrary data else it's defined in the specifications. • 6-7 = wLength, Length of the data to follow.

Returns

The number of returned bytes (in replyBuffer[]).

3.7.3.3 void writePin (volatile uint8_t * *port,* uint8_t *pinNbr,* uint8_t *pinStat*)

Pin-Setting-Function.

Sets the status of a specified pin (HIGH or LOW).

Parameters

port	Port the pin belongs to
pinNbr	Number of the pin
pinStat	Status of the pin (HIGH or LOW)

3.8 firmware/osccal.c File Reference

Sourcefile to osccal.h.

```
#include <avr/io.h>   #include <avr/eeprom.h>   #include
"./usbdrv/usbdrv.h"
```

Functions

- void calibrateOscillator (void)

  Calibrate the RC oscillator.
- void usbEventResetReady (void)

  Reset hook.
- void loadOldCallibartion (void)

  Loads the old calibration value for the internal rc oscillator.

3.8.1 Detailed Description

Sourcefile to osccal.h.

See also

osccal.h

Copyright: (c) 2008 by OBJECTIVE DEVELOPMENT Software GmbH License: GNU GPL v2 (see License.txt), GNU GPL v3 or proprietary (CommercialLicense.txt)

Date

2008-04-10

Author

Christian Starkjohann

Version

This Revision: $Id: osccal.h 762 2009-08-12 17:10:30Z cs

3.8.2 Function Documentation

3.8.2.1 void calibrateOscillator (void)

Calibrate the RC oscillator.

Our timing reference is the Start Of Frame signal (a single SE0 bit) repeating every millisecond immediately after a USB RESET. We first do a binary search for the OSCCAL value and then optimize this value with a neighboorhod search.

Note

This calibration algorithm may try OSCCAL values of up to 192 even if the optimum value is far below 192. It may therefore exceed the allowed clock frequency of the CPU in low voltage designs! You may replace this search algorithm with any other algorithm you like if you have additional constraints such as a maximum CPU clock. For version 5.x RC oscillators (those with a split range of 2x128 steps, e.g. ATTiny25, ATTiny45, ATTiny85), it may be useful to search for the optimum in both regions.
This function calibrates the RC oscillator so that the CPU runs at F_CPU. It MUST be called immediately after the end of a USB RESET condition! Disable all interrupts during the call! It is recommended that you store the resulting value in EEPROM so that a good guess value is available after the next reset.

3.8.2.2 void loadOldCallibartion (void)

Loads the old calibration value for the internal rc oscillator.

Loads old oscillator calibration value.

3.8.2.3 void usbEventResetReady (void)

Reset hook.

Necessary for the calibration via USB.

See also

calibrateOscillator

Returns

Your Mom!

3.9 firmware/osccal.h File Reference

Responsoble for the calibration of the interal RC oscillator.

Functions

- void calibrateOscillator (void)

  Calibrate the RC oscillator.
- void loadOldCallibartion (void)

  Loads old oscillator calibration value.

3.9.1 Detailed Description

Responsoble for the calibration of the interal RC oscillator. This module contains a function which calibrates the AVR's internal RC oscillator so that the CPU runs at F_CPU (F_CPU is a macro which must be defined when the module is compiled, best passed in the compiler command line).

The time reference is the USB frame clock of 1 kHz available immediately after a USB RESET condition. Timing is done by counting CPU cycles, so all interrupts must be disabled while the calibration runs. For low level timing measurements, usbMeasureFrameLength() is called. This function must be enabled in usbconfig.h by defining USB_CFG_HAVE_MEASURE_FRAME_LENGTH to 1. It is recommended to call calibrateOscillator() from the reset hook in usbconfig.h:

```
#ifndef __ASSEMBLER__
#include <avr/interrupt.h>  // for sei()
extern void calibrateOscillator(void);
#endif
#define USB_RESET_HOOK(resetStarts)  if(!resetStarts){cli();
      calibrateOscillator(); sei();}
```

This routine is an alternative to the continuous synchronization described in osctune.h.

3.9.2 Algorithm

calibrateOscillator() first does a binary search in the OSCCAL register for the best matching oscillator frequency. Then it does a next neighbor search to find the value with the lowest clock rate deviation. It is guaranteed to find the best match among neighboring values, but for version 5 oscillators (which have a discontinuous relationship between OSCCAL and frequency) a better match might be available in another OSCCAL region.

3.9.3 Limitations

This calibration algorithm may try OSCCAL values of up to 192 even if the optimum value is far below 192. It may therefore exceed the allowed clock frequency of the CPU in low voltage designs! Precision depends on the OSCCAL vs. frequency dependency of the oscillator. Typical precision for an ATMega168 (derived from the OSCCAL vs. F_RC diagram in the data sheet) should be in the range of 0.4%. Only the 12.8 MHz and 16.5 MHz versions of V-USB (with built-in receiver PLL) can tolerate this deviation! All other frequency modules require at least 0.2% precision.

Date

2008-04-10

Author

Christian Starkjohann

Version

This Revision: $Id: osccal.h 762 2009-08-12 17:10:30Z cs

3.9.4 Function Documentation

3.9.4.1 void calibrateOscillator (void)

Calibrate the RC oscillator.

Our timing reference is the Start Of Frame signal (a single SE0 bit) repeating every millisecond immediately after a USB RESET. We first do a binary search for the OSCCAL value and then optimize this value with a neighboorhod search.

Note

This calibration algorithm may try OSCCAL values of up to 192 even if the optimum value is far below 192. It may therefore exceed the allowed clock frequency of the CPU in low voltage designs! You may replace this search algorithm with any other algorithm you like if you have additional constraints such as a maximum CPU clock. For version 5.x RC oscillators (those with a split range of 2x128 steps, e.g. ATTiny25, ATTiny45, ATTiny85), it may be useful to search for the optimum in both regions.
This function calibrates the RC oscillator so that the CPU runs at F_CPU. It MUST be called immediately after the end of a USB RESET condition! Disable all interrupts during the call! It is recommended that you store the resulting value in EEPROM so that a good guess value is available after the next reset.

3.9.4.2 void loadOldCallibartion (void)

Loads old oscillator calibration value.

Returns

Sqrt(i)

3.10 firmware/requests.h File Reference

This header is shared between the firmware and the host software.

Defines

- #define CMD_GET 1

Command code for GET-Request.

- #define CMD_SET 2

  Command code for SET-Request.

3.10.1 Detailed Description

This header is shared between the firmware and the host software. It defines the USB request numbers used to communicate between the host and the device.

Date

31 Oct. 2010

Author

C. Bischko

Copyright

3.10.2 Define Documentation

3.10.2.1 #define CMD_GET 1

Command code for GET-Request.

3.10.2.2 #define CMD_SET 2

Command code for SET-Request.