DSP Controller

1.1

Generated by Doxygen 1.8.5

Mon Dec 2 2013 20:47:53

Contents

Chapter 1

Main Page

DSP Controller

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.1

DSPController is an interface device that expands the capability of the Analog Devices ADSP-21364 EZ-KIT Lite SHARC evaluation board. The evaluation board's human interface is limited by default. It contains 4 buttons and 8 leds.

The DSPController addresses this lack of functionality, and expands the evaluation board with

- 16x2 LCD display
- 2x8 led general purpose led bars
- 3 encoders
- 4 function buttons
- 5 arrow buttons
- a numeric pad, that can be used as 12 individual buttons
- 8 dip switches

The DSPController connects to the evaluation card via SPI. The two circuit are completely separated from each other. The separation is done by an Analog Devices ADUM isolator chip.

This document contains the documentation of the DSPController firmware itself. It doesn't include the HOST software.

Chapter 2

License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law:

that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

1. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

1. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components

(compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

1. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

2. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

3. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

4. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

1. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

2. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

1. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

1. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
2. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Main

Entry point and main control.

Modules

- LED bars

  Two general purpose led bars with 8-8 leds, mainly used for volume meter.

Files

- file includes.h
- file main.c
- file main.h

Macros

- #define F_CPU 16000000UL

Functions

- int main (void)
- void io_init ()
- void timer_init ()

Variables

- volatile uint8_t _led_l = 0
- volatile uint8_t _led_r = 0
- volatile uint8_t debounce [32]
- volatile uint16_t button_status [32]
- volatile uint8_t encoder_debounce [6]
- volatile int8_t encoder_counter [3]
- volatile uint8_t encoder_status [6]
- volatile uint8_t dip_status
- volatile uint8_t spi_state

- volatile uint8_t spi_transmit_A_not_B
- volatile uint8_t spi_transmit_pointer_A
- volatile uint8_t spi_transmit_pointer_B
- volatile uint8_t spi_transmit_buffer_A [40]
- volatile uint8_t spi_transmit_buffer_B [40]
- volatile uint8_t * spi_transmit_pointer_READ
- volatile uint8_t * spi_transmit_buffer_READ
- volatile uint8_t * spi_transmit_pointer_WRITE
- volatile uint8_t * spi_transmit_buffer_WRITE
- volatile uint8_t spi_receive_pointer
- volatile uint8_t spi_receive_buffer [36]
- volatile uint8_t spi_flag

5.1.1 Detailed Description

Entry point and main control.

5.1.2 Macro Definition Documentation

5.1.2.1 #define F_CPU 16000000UL

CPU clock frequency

Definition at line 22 of file includes.h.

5.1.3 Function Documentation

5.1.3.1 void io_init (void)

Initialize the AVR IO ports. It uses the macros defined in the BSP file.

Definition at line 88 of file main.c.

```
88                {
89     // shift registers
90     setOutput(IO_CLK);
91     setOutput(OUT_LATCH);
92     setOutput(IN_LOAD);
93     setOutput(OUT);
94
95     setInput(IN_1);
96     setInput(IN_2);
97     setInput(IN_3);
98     setInput(IN_4);
99
100     // encoders
101     setInputWPullup(E1_A);
102     setInputWPullup(E2_A);
103     setInputWPullup(E3_A);
104
105     setInputWPullup(E1_B);
106     setInputWPullup(E2_B);
107     setInputWPullup(E3_B);
108
109     // IN_LOAD is low active, set it high
110     setHigh(IN_LOAD);
111 }
```

5.1.3.2 int main (void)

Entry point of the firmware.

Definition at line 17 of file main.c.

```
17              {
18
19      // local string buffer for the LCD
20      char s[16];
21      // local cycle counter
22      int i = 0;
23
24      // init all of the modules
25      io_init();
26      input_init();
27      lcd_init();
28      timer_init();
29      usart_logger_init();
30      spi_init();
31
32
33      // write greetings text on the display
34      lcd_home();
35      sprintf(s," DSP Controller ");
36      lcd_writeString(s);
37      lcd_newLine();
38      sprintf(s,"      v1.1      ");
39      lcd_writeString(s);
40
41      // start the core engine
42      sei();
43
44
45      // infinite loop with LCD and LED handling
46      while(1) {
47
48          // if LED command was arrived via the spi, write it out atomically
49          if (spi_flag == SPI_FLAG_LED) {
50              cli();
51              setLed(spi_receive_buffer[1],
    spi_receive_buffer[0]);
52              spi_flag = SPI_FLAG_NONE;
53              sei();
54              refreshLeds();
55          }
56
57          // if TOP LCD command was arrived via the spi, write it out atomically
58          if (spi_flag == SPI_FLAG_LCD_TOP) {
59              cli();
60              i = 15;
61              // copy characters from the SPI buffer to the local buffer
62              do {
63                  s[i] = spi_receive_buffer[i];
64              } while (i--);
65              spi_flag = SPI_FLAG_NONE;
66              sei();
67              lcd_home();
68              lcd_writeString(s);
69          }
70
71          // if BOTTOM LCD command was arrived via the spi, write it out atomically
72          if (spi_flag == SPI_FLAG_LCD_BOTTOM) {
73              cli();
74              i = 15;
75              do {
76                  s[i] = spi_receive_buffer[i];
77              } while (i--);
78              spi_flag = SPI_FLAG_NONE;
79              sei();
80              lcd_newLine();
81              lcd_writeString(s);
82          }
83      }
84      return 0;
85  }
```

5.1.3.3 void timer_init (void)

Initialize Timer0 to Output Compare Match A Interrupt @ 0.5 ms. This interrupt will trigger all input processing algorithm.

Definition at line 114 of file main.c.

```
114                 {
115      // Triggers an interrupt in each 0.5 miliseconds.
116      // enable Timer/Counter0 Output Compare Match A Interrupt
```

```
117     TIMSK0 = (1<<OCIE0A);
118
119     // set CTC mode
120     TCCR0A = (1<<WGM01) | (0<<WGM00);
121
122     // prescaler: 64, CTC 125 = 0.5ms interrupts
123     OCR0A = 125; // 0.5ms @ div64
124
125     // set prescaler to 64
126     TCCR0B = (0<<CS02) | (1<<CS01) | (1<<CS00);  // div 64
127 }
```

5.1.4 Variable Documentation

5.1.4.1 volatile uint8_t _led_l = 0

Global variable that holds the left led bar's current value.

Definition at line 20 of file main.h.

5.1.4.2 volatile uint8_t _led_r = 0

Global variable that holds the right led bar's current value..

Definition at line 21 of file main.h.

5.1.4.3 volatile uint16_t button_status[32]

Global array of variables that store the buttons' status information.

Definition at line 25 of file main.h.

5.1.4.4 volatile uint8_t debounce[32]

Global array of variables that act as a debounce registers fr the buttons.

Definition at line 24 of file main.h.

5.1.4.5 volatile uint8_t dip_status

Global variable that contains the current dip status.

Definition at line 31 of file main.h.

5.1.4.6 volatile int8_t encoder_counter[3]

Global array that holds the actual encoder increments.

Definition at line 28 of file main.h.

5.1.4.7 volatile uint8_t encoder_debounce[6]

Global array of variables that act as a debounce registers for the encoders.

Definition at line 27 of file main.h.

5.1.4.8 volatile uint8_t encoder_status[6]

Global array that holds the encoders' status information.

Definition at line 29 of file main.h.

5.1.4.9 volatile uint8_t spi_flag

Global variable that .

Definition at line 49 of file main.h.

5.1.4.10 volatile uint8_t spi_receive_buffer[36]

Global variable that .

Definition at line 48 of file main.h.

5.1.4.11 volatile uint8_t spi_receive_pointer

Global variable that .

Definition at line 47 of file main.h.

5.1.4.12 volatile uint8_t spi_state

Global variable that holds the actual SPI state.

Definition at line 34 of file main.h.

5.1.4.13 volatile uint8_t spi_transmit_A_not_B

Global variable that decides which SPI buffer will be the next readable buffer.

Definition at line 36 of file main.h.

5.1.4.14 volatile uint8_t spi_transmit_buffer_A[40]

Global variable that .

Definition at line 39 of file main.h.

5.1.4.15 volatile uint8_t spi_transmit_buffer_B[40]

Global variable that .

Definition at line 40 of file main.h.

5.1.4.16 volatile uint8_t* spi_transmit_buffer_READ

Global variable that .

Definition at line 43 of file main.h.

5.1.4.17 volatile uint8_t* spi_transmit_buffer_WRITE

Global variable that .

Definition at line 45 of file main.h.

5.1.4.18 volatile uint8_t spi_transmit_pointer_A

Global variable that .

Definition at line 37 of file main.h.

5.1.4.19 volatile uint8_t spi_transmit_pointer_B

Global variable that .

Definition at line 38 of file main.h.

5.1.4.20 volatile uint8_t* spi_transmit_pointer_READ

Global variable that .

Definition at line 42 of file main.h.

5.1.4.21 volatile uint8_t* spi_transmit_pointer_WRITE

Global variable that .

Definition at line 44 of file main.h.

5.2 Debug tools

Tools for debugging the project.

Modules

- USART Logger

  A simple lightweight console debug tool that redirects standard io stream to the USART hardware.

5.2.1 Detailed Description

Tools for debugging the project. The module is included to the code in this documentation but it should be removed int the final release.

5.3 Board Support Package

Low level layer that hides the hardware from the software. It implements a simple to use macro system, that is easy to expand.

Files

- file bsp.h

Macros

- #define IO_CLK IO_CLK
- #define OUT_LATCH OUT_LATCH
- #define IN_LOAD IN_LOAD
- #define OUT OUT
- #define IN_1 IN_1
- #define IN_2 IN_2
- #define IN_3 IN_3
- #define IN_4 IN_4
- #define E1_A E1_A
- #define E1_B E1_B
- #define E2_A E2_A
- #define E2_B E2_B
- #define E3_A E3_A
- #define E3_B E3_B
- #define MOSI MOSI
- #define MISO MISO
- #define SCK SCK
- #define SS SS
- #define MOSI_DDR DDRB
- #define MOSI_PORT PORTB
- #define MOSI_PIN PINB
- #define MOSI_NAME PB3
- #define MISO_DDR DDRB
- #define MISO_PORT PORTB
- #define MISO_PIN PINB
- #define MISO_NAME PB4
- #define SS_DDR DDRB
- #define SS_PORT PORTB
- #define SS_PIN PINB
- #define SS_NAME PB2
- #define SCK_DDR DDRB
- #define SCK_PORT PORTB
- #define SCK_PIN PINB
- #define SCK_NAME PB5
- #define IO_CLK_DDR DDRC
- #define IO_CLK_PORT PORTC
- #define IO_CLK_PIN PINC
- #define IO_CLK_NAME PC0
- #define OUT_LATCH_DDR DDRC
- #define OUT_LATCH_PORT PORTC
- #define OUT_LATCH_PIN PINC
- #define OUT_LATCH_NAME PC1

- #define IN_LOAD_DDR DDRC
- #define IN_LOAD_PORT PORTC
- #define IN_LOAD_PIN PINC
- #define IN_LOAD_NAME PC2
- #define OUT_DDR DDRC
- #define OUT_PORT PORTC
- #define OUT_PIN PINC
- #define OUT_NAME PC3
- #define IN_1_DDR DDRC
- #define IN_1_PORT PORTC
- #define IN_1_PIN PINC
- #define IN_1_NAME PC4
- #define IN_2_DDR DDRC
- #define IN_2_PORT PORTC
- #define IN_2_PIN PINC
- #define IN_2_NAME PC5
- #define IN_3_DDR DDRB
- #define IN_3_PORT PORTB
- #define IN_3_PIN PINB
- #define IN_3_NAME PB0
- #define IN_4_DDR DDRB
- #define IN_4_PORT PORTB
- #define IN_4_PIN PINB
- #define IN_4_NAME PB1
- #define E1_A_DDR DDRD
- #define E1_A_PORT PORTD
- #define E1_A_PIN PIND
- #define E1_A_NAME PD3
- #define E1_B_DDR DDRD
- #define E1_B_PORT PORTD
- #define E1_B_PIN PIND
- #define E1_B_NAME PD2
- #define E2_A_DDR DDRD
- #define E2_A_PORT PORTD
- #define E2_A_PIN PIND
- #define E2_A_NAME PD5
- #define E2_B_DDR DDRD
- #define E2_B_PORT PORTD
- #define E2_B_PIN PIND
- #define E2_B_NAME PD4
- #define E3_A_DDR DDRD
- #define E3_A_PORT PORTD
- #define E3_A_PIN PIND
- #define E3_A_NAME PD6
- #define E3_B_DDR DDRD
- #define E3_B_PORT PORTD
- #define E3_B_PIN PIND
- #define E3_B_NAME PD7
- #define IN_1_A 0x01
- #define IN_1_B 0x02
- #define IN_1_C 0x04
- #define IN_1_D 0x08
- #define IN_1_E 0x10
- #define IN_1_F 0x20
- #define IN_1_G 0x40

- #define IN_1_H 0x80
- #define IN_2_A 0x01
- #define IN_2_B 0x02
- #define IN_2_C 0x04
- #define IN_2_D 0x08
- #define IN_2_E 0x10
- #define IN_2_F 0x20
- #define IN_2_G 0x40
- #define IN_2_H 0x80
- #define IN_3_A 0x01
- #define IN_3_B 0x02
- #define IN_3_C 0x04
- #define IN_3_D 0x08
- #define IN_3_E 0x10
- #define IN_3_F 0x20
- #define IN_3_G 0x40
- #define IN_3_H 0x80
- #define IN_4_A 0x01
- #define IN_4_B 0x02
- #define IN_4_C 0x04
- #define IN_4_D 0x08
- #define IN_4_E 0x10
- #define IN_4_F 0x20
- #define IN_4_G 0x40
- #define IN_4_H 0x80
- #define DIP_1 IN_1_A
- #define DIP_2 IN_1_B
- #define DIP_3 IN_1_C
- #define DIP_4 IN_1_D
- #define DIP_5 IN_1_E
- #define DIP_6 IN_1_F
- #define DIP_7 IN_1_G
- #define DIP_8 IN_1_H
- #define E1 IN_2_A
- #define E2 IN_2_B
- #define E3 IN_2_C
- #define F1 IN_2_D
- #define F2 IN_2_E
- #define F3 IN_2_F
- #define F4 IN_2_G
- #define A1 IN_3_D
- #define A2 IN_2_H
- #define A3 IN_3_B
- #define A4 IN_3_C
- #define A5 IN_3_A
- #define N1 IN_4_A
- #define N2 IN_4_B
- #define N3 IN_4_C
- #define N4 IN_4_D
- #define N5 IN_4_E
- #define N6 IN_4_F
- #define N7 IN_4_G
- #define N8 IN_4_H
- #define N9 IN_3_H
- #define N10 IN_3_E

- #define N11 IN_3_F
- #define N12 IN_3_G
- #define output(ddr, name) ((ddr) |= (1 << (name)))
- #define input(ddr, name) ((ddr) &= ~(1 << (name)))
- #define setOutput(name) output(name##_DDR,name##_NAME)
- #define setInput(name) input(name##_DDR,name##_NAME)
- #define setInputWPullup(name)
- #define toggle(pin, name) ((pin) |= (1 << (name)))
- #define low(port, name) ((port) &= ~(1 << (name)))
- #define high(port, name) ((port) |= (1 << (name)))
- #define setLow(name) low(name##_PORT,name##_NAME)
- #define setHigh(name) high(name##_PORT,name##_NAME)
- #define read(pin, name) (((pin) & (1<<name)) >> name)
- #define readValue(name) read(name##_PIN,name##_NAME)
- #define pulse(name)
- #define negativePulse(name)
- #define setLed(L, R)

5.3.1 Detailed Description

Low level layer that hides the hardware from the software. It implements a simple to use macro system, that is easy to expand.

5.3.2 Macro Definition Documentation

5.3.2.1 #define A1 IN_3_D

Association between input source and shift register pin for the ARROW BUTTONS.

Definition at line 236 of file bsp.h.

5.3.2.2 #define A2 IN_2_H

Association between input source and shift register pin for the ARROW BUTTONS.

Definition at line 237 of file bsp.h.

5.3.2.3 #define A3 IN_3_B

Association between input source and shift register pin for the ARROW BUTTONS.

Definition at line 238 of file bsp.h.

5.3.2.4 #define A4 IN_3_C

Association between input source and shift register pin for the ARROW BUTTONS.

Definition at line 239 of file bsp.h.

5.3.2.5 #define A5 IN_3_A

Association between input source and shift register pin for the ARROW BUTTONS.

Definition at line 240 of file bsp.h.

5.3.2.6 #define DIP_1 IN_1_A

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 203 of file bsp.h.

5.3.2.7 #define DIP_2 IN_1_B

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 204 of file bsp.h.

5.3.2.8 #define DIP_3 IN_1_C

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 205 of file bsp.h.

5.3.2.9 #define DIP_4 IN_1_D

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 206 of file bsp.h.

5.3.2.10 #define DIP_5 IN_1_E

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 207 of file bsp.h.

5.3.2.11 #define DIP_6 IN_1_F

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 208 of file bsp.h.

5.3.2.12 #define DIP_7 IN_1_G

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 209 of file bsp.h.

5.3.2.13 #define DIP_8 IN_1_H

Association between input source and shift register pin for the DIP_SWITCH.

Definition at line 210 of file bsp.h.

5.3.2.14 #define E1 IN_2_A

Association between input source and shift register pin for the encoder buttons.

Definition at line 217 of file bsp.h.

5.3.2.15 #define E1_A E1_A

First encoder's A signal.

Definition at line 28 of file bsp.h.

5.3.2.16 #define E1_A_DDR DDRD

Data Direction Register for E1_A.

Definition at line 115 of file bsp.h.

5.3.2.17 #define E1_A_NAME PD3

Pin name for E1_A.

Definition at line 118 of file bsp.h.

5.3.2.18 #define E1_A_PIN PIND

Pin Register for E1_A.

Definition at line 117 of file bsp.h.

5.3.2.19 #define E1_A_PORT PORTD

Port Register for E1_A.

Definition at line 116 of file bsp.h.

5.3.2.20 #define E1_B E1_B

First encoder's B signal.

Definition at line 29 of file bsp.h.

5.3.2.21 #define E1_B_DDR DDRD

Data Direction Register for E1_B.

Definition at line 120 of file bsp.h.

5.3.2.22 #define E1_B_NAME PD2

Pin name for E1_B.

Definition at line 123 of file bsp.h.

5.3.2.23 #define E1_B_PIN PIND

Pin Register for E1_B.

Definition at line 122 of file bsp.h.

5.3.2.24 #define E1_B_PORT PORTD

Port Register for E1_B.

Definition at line 121 of file bsp.h.

5.3.2.25 #define E2 IN_2_B

Association between input source and shift register pin for the encoder buttons.

Definition at line 218 of file bsp.h.

5.3.2.26 #define E2_A E2_A

Second encoder's A signal.

Definition at line 30 of file bsp.h.

5.3.2.27 #define E2_A_DDR DDRD

Data Direction Register for E2_A.

Definition at line 125 of file bsp.h.

5.3.2.28 #define E2_A_NAME PD5

Pin name for E2_A.

Definition at line 128 of file bsp.h.

5.3.2.29 #define E2_A_PIN PIND

Pin Register for E2_A.

Definition at line 127 of file bsp.h.

5.3.2.30 #define E2_A_PORT PORTD

Port Register for E2_A.

Definition at line 126 of file bsp.h.

5.3.2.31 #define E2_B E2_B

Second encoder's B signal.

Definition at line 31 of file bsp.h.

5.3.2.32 #define E2_B_DDR DDRD

Data Direction Register for E2_B.

Definition at line 130 of file bsp.h.

5.3.2.33 #define E2_B_NAME PD4

Pin name for E2_B.

Definition at line 133 of file bsp.h.

5.3.2.34 #define E2_B_PIN PIND

Pin Register for E2_B.

Definition at line 132 of file bsp.h.

5.3.2.35 #define E2_B_PORT PORTD

Port Register for E2_B.

Definition at line 131 of file bsp.h.

5.3.2.36 #define E3 IN_2_C

Association between input source and shift register pin for the encoder buttons.

Definition at line 219 of file bsp.h.

5.3.2.37 #define E3_A E3_A

Third encoder's A signal.

Definition at line 32 of file bsp.h.

5.3.2.38 #define E3_A_DDR DDRD

Data Direction Register for E3_A.

Definition at line 135 of file bsp.h.

5.3.2.39 #define E3_A_NAME PD6

Pin name for E3_A.

Definition at line 138 of file bsp.h.

5.3.2.40 #define E3_A_PIN PIND

Pin Register for E3_A.

Definition at line 137 of file bsp.h.

5.3.2.41 #define E3_A_PORT PORTD

Port Register for E3_A.

Definition at line 136 of file bsp.h.

5.3.2.42 #define E3_B E3_B

Third encoder's B signal.

Definition at line 33 of file bsp.h.

5.3.2.43 #define E3_B_DDR DDRD

Data Direction Register for E3_B.

Definition at line 140 of file bsp.h.

5.3.2.44 #define E3_B_NAME PD7

Pin name for E3_B.

Definition at line 143 of file bsp.h.

5.3.2.45 #define E3_B_PIN PIND

Pin Register for E3_B.

Definition at line 142 of file bsp.h.

5.3.2.46 #define E3_B_PORT PORTD

Port Register for E3_B.

Definition at line 141 of file bsp.h.

5.3.2.47 #define F1 IN_2_D

Association between input source and shift register pin for the FUNCTION BUTTONS.

Definition at line 226 of file bsp.h.

5.3.2.48 #define F2 IN_2_E

Association between input source and shift register pin for the FUNCTION BUTTONS.

Definition at line 227 of file bsp.h.

5.3.2.49 #define F3 IN_2_F

Association between input source and shift register pin for the FUNCTION BUTTONS.

Definition at line 228 of file bsp.h.

5.3.2.50 #define F4 IN_2_G

Association between input source and shift register pin for the FUNCTION BUTTONS.

Definition at line 229 of file bsp.h.

5.3.2.51 #define high(*port, name*) ((port) |= (1 << (name)))

Low level macro that sets the given pin to high logical state.

Parameters

in	*port*	The PORTX register for the given pin.
in	*name*	The name of the given pin.

Definition at line 329 of file bsp.h.

5.3.2.52 #define IN_1 IN_1

Input serial data line from the first input shift register.

Definition at line 24 of file bsp.h.

5.3.2.53 #define IN_1_A 0x01

Association for shift register IN_1.

Definition at line 150 of file bsp.h.

5.3.2.54 #define IN_1_B 0x02

Association for shift register IN_1.

Definition at line 151 of file bsp.h.

5.3.2.55 #define IN_1_C 0x04

Association for shift register IN_1.

Definition at line 152 of file bsp.h.

5.3.2.56 #define IN_1_D 0x08

Association for shift register IN_1.

Definition at line 153 of file bsp.h.

5.3.2.57 #define IN_1_DDR DDRC

Data Direction Register for IN_1.

Definition at line 90 of file bsp.h.

5.3.2.58 #define IN_1_E 0x10

Association for shift register IN_1.

Definition at line 154 of file bsp.h.

5.3.2.59 #define IN_1_F 0x20

Association for shift register IN_1.

Definition at line 155 of file bsp.h.

5.3.2.60 #define IN_1_G 0x40

Association for shift register IN_1.

Definition at line 156 of file bsp.h.

5.3.2.61 #define IN_1_H 0x80

Association for shift register IN_1.

Definition at line 157 of file bsp.h.

5.3.2.62 #define IN_1_NAME PC4

Pin name for IN_1.

Definition at line 93 of file bsp.h.

5.3.2.63 #define IN_1_PIN PINC

Pin Register for IN_1.

Definition at line 92 of file bsp.h.

5.3.2.64 #define IN_1_PORT PORTC

Port Register for IN_1.

Definition at line 91 of file bsp.h.

5.3.2.65 #define IN_2 IN_2

Input serial data line from the second input shift register.

Definition at line 25 of file bsp.h.

5.3.2.66 #define IN_2_A 0x01

Association for shift register IN_2.

Definition at line 163 of file bsp.h.

5.3.2.67 #define IN_2_B 0x02

Association for shift register IN_2.

Definition at line 164 of file bsp.h.

5.3.2.68 #define IN_2_C 0x04

Association for shift register IN_2.

Definition at line 165 of file bsp.h.

5.3.2.69 #define IN_2_D 0x08

Association for shift register IN_2.

Definition at line 166 of file bsp.h.

5.3.2.70 #define IN_2_DDR DDRC

Data Direction Register for IN_2.

Definition at line 95 of file bsp.h.

5.3.2.71 #define IN_2_E 0x10

Association for shift register IN_2.

Definition at line 167 of file bsp.h.

5.3.2.72 #define IN_2_F 0x20

Association for shift register IN_2.

Definition at line 168 of file bsp.h.

5.3.2.73 #define IN_2_G 0x40

Association for shift register IN_2.

Definition at line 169 of file bsp.h.

5.3.2.74 #define IN_2_H 0x80

Association for shift register IN_2.

Definition at line 170 of file bsp.h.

5.3.2.75 #define IN_2_NAME PC5

Pin name for IN_2.

Definition at line 98 of file bsp.h.

5.3.2.76 #define IN_2_PIN PINC

Pin Register for IN_2.

Definition at line 97 of file bsp.h.

5.3.2.77 #define IN_2_PORT PORTC

Port Register for IN_2.

Definition at line 96 of file bsp.h.

5.3.2.78 #define IN_3 IN_3

Input serial data line from the third input shift register.

Definition at line 26 of file bsp.h.

5.3.2.79 #define IN_3_A 0x01

Association for shift register IN_3.

Definition at line 176 of file bsp.h.

5.3.2.80 #define IN_3_B 0x02

Association for shift register IN_3.

Definition at line 177 of file bsp.h.

5.3.2.81 #define IN_3_C 0x04

Association for shift register IN_3.

Definition at line 178 of file bsp.h.

5.3.2.82 #define IN_3_D 0x08

Association for shift register IN_3.

Definition at line 179 of file bsp.h.

5.3.2.83 #define IN_3_DDR DDRB

Data Direction Register for IN_3.

Definition at line 100 of file bsp.h.

5.3.2.84 #define IN_3_E 0x10

Association for shift register IN_3.

Definition at line 180 of file bsp.h.

5.3.2.85 #define IN_3_F 0x20

Association for shift register IN_3.

Definition at line 181 of file bsp.h.

5.3.2.86 #define IN_3_G 0x40

Association for shift register IN_3.

Definition at line 182 of file bsp.h.

5.3.2.87 #define IN_3_H 0x80

Association for shift register IN_3.

Definition at line 183 of file bsp.h.

5.3.2.88 #define IN_3_NAME PB0

Pin name for IN_3.

Definition at line 103 of file bsp.h.

5.3.2.89 #define IN_3_PIN PINB

Pin Register for IN_3.

Definition at line 102 of file bsp.h.

5.3.2.90 #define IN_3_PORT PORTB

Port Register for IN_3.

Definition at line 101 of file bsp.h.

5.3.2.91 #define IN_4 IN_4

Input serial data line from the fourth input shift register.

Definition at line 27 of file bsp.h.

5.3.2.92 #define IN_4_A 0x01

Association for shift register IN_4.

Definition at line 189 of file bsp.h.

5.3.2.93 #define IN_4_B 0x02

Association for shift register IN_4.

Definition at line 190 of file bsp.h.

5.3.2.94 #define IN_4_C 0x04

Association for shift register IN_4.

Definition at line 191 of file bsp.h.

5.3.2.95 #define IN_4_D 0x08

Association for shift register IN_4.

Definition at line 192 of file bsp.h.

5.3.2.96 #define IN_4_DDR DDRB

Data Direction Register for IN_4.

Definition at line 105 of file bsp.h.

5.3.2.97 #define IN_4_E 0x10

Association for shift register IN_4.

Definition at line 193 of file bsp.h.

5.3.2.98 #define IN_4_F 0x20

Association for shift register IN_4.

Definition at line 194 of file bsp.h.

5.3.2.99 #define IN_4_G 0x40

Association for shift register IN_4.

Definition at line 195 of file bsp.h.

5.3.2.100 #define IN_4_H 0x80

Association for shift register IN_4.

Definition at line 196 of file bsp.h.

5.3.2.101 #define IN_4_NAME PB1

Pin name for IN_4.

Definition at line 108 of file bsp.h.

5.3.2.102 #define IN_4_PIN PINB

Pin Register for IN_4.

Definition at line 107 of file bsp.h.

5.3.2.103 #define IN_4_PORT PORTB

Port Register for IN_4.

Definition at line 106 of file bsp.h.

5.3.2.104 #define IN_LOAD IN_LOAD

Load signal for the input shift registers.

Definition at line 22 of file bsp.h.

5.3.2.105 #define IN_LOAD_DDR DDRC

Data Direction Register for IN_LOAD.

Definition at line 80 of file bsp.h.

5.3.2.106 #define IN_LOAD_NAME PC2

Pin name for IN_LOAD.

Definition at line 83 of file bsp.h.

5.3.2.107 #define IN_LOAD_PIN PINC

Pin Register for IN_LOAD.

Definition at line 82 of file bsp.h.

5.3.2.108 #define IN_LOAD_PORT PORTC

Port Register for IN_LOAD.

Definition at line 81 of file bsp.h.

5.3.2.109 #define input(*ddr, name*) ((ddr) &= ~(1 << (name)))

Low level macro that configures the given pin as an input.

Parameters

in	*ddr*	DDR register that corresponds to the given pin.
in	*name*	The name of the given pin.

Definition at line 277 of file bsp.h.

5.3.2.110 #define IO_CLK IO_CLK

Clk line for all of the shift registers.

Definition at line 20 of file bsp.h.

5.3.2.111 #define IO_CLK_DDR DDRC

Data Direction Register for IO_CLK.

Definition at line 70 of file bsp.h.

5.3.2.112 #define IO_CLK_NAME PC0

Pin name for IO_CLK.

Definition at line 73 of file bsp.h.

5.3.2.113 #define IO_CLK_PIN PINC

Pin Register for IO_CLK.

Definition at line 72 of file bsp.h.

5.3.2.114 #define IO_CLK_PORT PORTC

Port Register for IO_CLK.

Definition at line 71 of file bsp.h.

5.3.2.115 #define low(*port, name*) ((port) &= ∼(1 << (name)))

Low level macro that sets the given pin to low logical state.

Parameters

in	*port*	The PORTX register for the given pin.
in	*name*	The name of the given pin.

Definition at line 322 of file bsp.h.

5.3.2.116 #define MISO MISO

SPI MISO signal.

Definition at line 35 of file bsp.h.

5.3.2.117 #define MISO_DDR DDRB

Data Direction Register for MISO.

Definition at line 50 of file bsp.h.

5.3.2.118 #define MISO_NAME PB4

Pin name for MISO.

Definition at line 53 of file bsp.h.

5.3.2.119 #define MISO_PIN PINB

Pin Register for MISO.

Definition at line 52 of file bsp.h.

5.3.2.120 #define MISO_PORT PORTB

Port Register for MISO.

Definition at line 51 of file bsp.h.

5.3.2.121 #define MOSI MOSI

SPI MOSI signal.

Definition at line 34 of file bsp.h.

5.3.2.122 #define MOSI_DDR DDRB

Data Direction Register for MOSI.

Definition at line 45 of file bsp.h.

5.3.2.123 #define MOSI_NAME PB3

Pin name for MOSI.

Definition at line 48 of file bsp.h.

5.3.2.124 #define MOSI_PIN PINB

Pin Register for MOSI.

Definition at line 47 of file bsp.h.

5.3.2.125 #define MOSI_PORT PORTB

Port Register for MOSI.

Definition at line 46 of file bsp.h.

5.3.2.126 #define N1 IN_4_A

Association between input source and shift register pin for the NUMPAD.

Definition at line 247 of file bsp.h.

5.3.2.127 #define N10 IN_3_E

Association between input source and shift register pin for the NUMPAD.

Definition at line 256 of file bsp.h.

5.3.2.128 #define N11 IN_3_F

Association between input source and shift register pin for the NUMPAD.

Definition at line 257 of file bsp.h.

5.3.2.129 #define N12 IN_3_G

Association between input source and shift register pin for the NUMPAD.

Definition at line 258 of file bsp.h.

5.3.2.130 #define N2 IN_4_B

Association between input source and shift register pin for the NUMPAD.

Definition at line 248 of file bsp.h.

5.3.2.131 #define N3 IN_4_C

Association between input source and shift register pin for the NUMPAD.

Definition at line 249 of file bsp.h.

5.3.2.132 #define N4 IN_4_D

Association between input source and shift register pin for the NUMPAD.

Definition at line 250 of file bsp.h.

5.3.2.133 #define N5 IN_4_E

Association between input source and shift register pin for the NUMPAD.

Definition at line 251 of file bsp.h.

5.3.2.134 #define N6 IN_4_F

Association between input source and shift register pin for the NUMPAD.

Definition at line 252 of file bsp.h.

5.3.2.135 #define N7 IN_4_G

Association between input source and shift register pin for the NUMPAD.

Definition at line 253 of file bsp.h.

5.3.2.136 #define N8 IN_4_H

Association between input source and shift register pin for the NUMPAD.

Definition at line 254 of file bsp.h.

5.3.2.137 #define N9 IN_3_H

Association between input source and shift register pin for the NUMPAD.

Definition at line 255 of file bsp.h.

5.3.2.138 #define negativePulse(*name*)

Value:

```
do { \
            setLow(name); \
            setHigh(name); \
        } while (0)
```

It pulses negatively the given pin. Pulse characteristic: HIGH - LOW - HIGH

Parameters

in	*name*	The name of the given pin.

Definition at line 380 of file bsp.h.

5.3.2.139 #define OUT OUT

Serial data out for the output shift registers.

Definition at line 23 of file bsp.h.

5.3.2.140 #define OUT_DDR DDRC

Data Direction Register for OUT.

Definition at line 85 of file bsp.h.

5.3.2.141 #define OUT_LATCH OUT_LATCH

Latch signal for the output shift registers.

Definition at line 21 of file bsp.h.

5.3.2.142 #define OUT_LATCH_DDR DDRC

Data Direction Register for OUT_LATCH.

Definition at line 75 of file bsp.h.

5.3.2.143 #define OUT_LATCH_NAME PC1

Pin name for OUT_LATCH.

Definition at line 78 of file bsp.h.

5.3.2.144 #define OUT_LATCH_PIN PINC

Pin Register for OUT_LATCH.

Definition at line 77 of file bsp.h.

5.3.2.145 #define OUT_LATCH_PORT PORTC

Port Register for OUT_LATCH.

Definition at line 76 of file bsp.h.

5.3.2.146 #define OUT_NAME PC3

Pin name for OUT.

Definition at line 88 of file bsp.h.

5.3.2.147 #define OUT_PIN PINC

Pin Register for OUT.

Definition at line 87 of file bsp.h.

5.3.2.148 #define OUT_PORT PORTC

Port Register for OUT.

Definition at line 86 of file bsp.h.

5.3.2.149 #define output(*ddr, name*) ((ddr) |= (1 << (name)))

Low level macro that configures the given pin as an output.

Parameters

in	*ddr*	DDR register that corresponds to the given pin.
in	*name*	The name of the given pin.

Definition at line 270 of file bsp.h.

5.3.2.150 #define pulse(*name*)

Value:

```
do { \
            setHigh(name); \
            setLow(name); \
        } while (0)
```

It pulses the given pin. Pulse characteristic: LOW - HIGH - LOW

Parameters

in	*name*	The name of the given pin.

Definition at line 370 of file bsp.h.

5.3.2.151 #define read(*pin, name*) (((pin) & (1<<name)) >> name)

Low level macro that reads the logical value of the given pin.

Parameters

in	*pin*	PINX register corresponding to the given pin.
in	*name*	The name of the given pin.

Definition at line 353 of file bsp.h.

5.3.2.152 #define readValue(*name*) read(name##_PIN,name##_NAME)

Higher level read macro that reads a pin's logical value.

Parameters

in	*name*	The name of the given pin.

Definition at line 359 of file bsp.h.

5.3.2.153 #define SCK SCK

SPI SCK signal.

Definition at line 36 of file bsp.h.

5.3.2.154 #define SCK_DDR DDRB

Data Direction Register for SCK.

Definition at line 60 of file bsp.h.

5.3.2.155 #define SCK_NAME PB5

Pin name for SCK.

Definition at line 63 of file bsp.h.

5.3.2.156 #define SCK_PIN PINB

Pin Register for SCK.

Definition at line 62 of file bsp.h.

5.3.2.157 #define SCK_PORT PORTB

Port Register for SCK.

Definition at line 61 of file bsp.h.

5.3.2.158 #define setHigh(*name*) high(name##_PORT,name##_NAME)

Higher level macro for setting the given pin state to high logic level.

Parameters

in	*name*	The name of the given pin.

Definition at line 341 of file bsp.h.

5.3.2.159 #define setInput(*name*) input(name##_DDR,name##_NAME)

Higher level macro to configure the given pin to input. It uses the low level macro to do this.

Parameters

in	*name*	The name of the given pin.

Definition at line 291 of file bsp.h.

5.3.2.160 #define setInputWPullup(*name*)

Value:

```
do { \
            output(name##_DDR,name##_NAME); \
            (name##_PORT) |= (1 << (name##_NAME)); \
        } while (0)
```

Higher level macro to configure the given pin to input and it turns on it's internal pullup resistor. It uses the low level macro to do this.

Parameters

in	*name*	The name of the given pin.

Definition at line 299 of file bsp.h.

5.3.2.161 #define setLed(*L*, *R*)

Value:

```
do { \
            _led_l = L; \
            _led_r = R; \
        } while (0)
```

This macro sets the two led bar's global variables that hold the current led bar configurations.

Parameters

in	*L*	Values for the left led bar.
in	*R*	Values for the right led bar.

Definition at line 397 of file bsp.h.

5.3.2.162 #define setLow(*name*) low(name##_PORT,name##_NAME)

Higher level macro for setting the given pin state to low logic level.

Parameters

in	*name*	The name of the given pin.

Definition at line 335 of file bsp.h.

5.3.2.163 #define setOutput(*name*) output(name##_DDR,name##_NAME)

Higher level macro to configure the given pin to output. It uses the low level macro to do this.

Parameters

in	*name*	The name of the given pin.

Definition at line 284 of file bsp.h.

5.3.2.164 #define SS SS

SPI SS signal.

Definition at line 37 of file bsp.h.

5.3.2.165 #define SS_DDR DDRB

Data Direction Register for SS.

Definition at line 55 of file bsp.h.

5.3.2.166 #define SS_NAME PB2

Pin name for SS.

Definition at line 58 of file bsp.h.

5.3.2.167 #define SS_PIN PINB

Pin Register for SS.

Definition at line 57 of file bsp.h.

5.3.2.168 #define SS_PORT PORTB

Port Register for SS.

Definition at line 56 of file bsp.h.

5.3.2.169 #define toggle(*pin, name*) ((pin) |= (1 << (name)))

Low level macro that toggles a pin based on the AVR IO architecture's hardware XOR feature.

Parameters

in	*pin*	The PINX register for the given pin.
in	*name*	The name of the given pin.

Definition at line 315 of file bsp.h.

5.4 Interfaces

Interface modules.

Modules

- Inputs

  All of the input interface modules.
- Outputs

  All of the output interface modules that drives the interface, including the LCD display and the LED bars.

5.4.1 Detailed Description

Interface modules.

5.5 Inputs

All of the input interface modules.

Modules

- Buttons

  All the buttons of the DSP controller are driven by shift registers. These shift registers are asked periodically for new data. In every period new data runs through a debouncer algorithm that generates events, and these events are stored in a buffer, from which the spi communication program reads them out and sends to the HOST system.
- Encoders

  The operation of the encoders is very similar to the buttons. But instead of using shift registers, it's actual value are readed out directly with some IO pins of the AVR microcontroller. These values processed through the debouncer algorithm, and than based on the result, the corresponding registers that counts the encoder's rotation are updated.

Files

- file input.c
- file input.h

Functions

- void input_init ()
- ISR (TIMER0_COMPA_vect)

5.5.1 Detailed Description

All of the input interface modules.

5.5.2 Function Documentation

5.5.2.1 void input_init ( )

Function that initializes the registers for the debounce algorithm.

Definition at line 14 of file input.c.

```
14                  {
15     uint8_t i = 31;
16
17     // zero out the button registers
18     do {
19         debounce[i] = 0;
20         button_status[i] = 0;
21     } while (i--);
22
23     // zero out encoder debounce register
24     // set encoder status registers:
25     //      index 0, 2, 4 will be masters -> A signals
26     //      all previous and actual value is 1 to prevent false triggering at boot time
27     i=5;
28     do {
29         encoder_debounce[i] = 0;
30         encoder_status[i] = i%2 ? 0xc0 : 0xe0;
31     } while (i--);
32
33     // zero out the encoder counters
34     encoder_counter[0] = 0;
35     encoder_counter[1] = 0;
36     encoder_counter[2] = 0;
37
38     // zero out dip_status
```

```
39     dip_status = 0;
40 }
```

5.5.2.2 ISR (TIMER0_COMPA_vect)

This ISR is provided the Timer Tick event that schedules the debouncing and processing jobs.

Definition at line 45 of file input.c.

```
45                       {
46
47     uint8_t i = 7;
48     uint8_t p = 0;
49
50     // the buttons need to be processed every second tick, we use a simple blocking variable
51     static uint8_t blocker = 0;
52
53     // if there is a change in one of the dip switches, that are processed the same way as
54     // the other buttons, toggle this flag, and at the end of the algorithm all of the dip
55     // switches will be refreshed
56     uint8_t dip_update = 0;
57
58     // turn on interrupts because when an interrupts occurs, the AVR turns of the global
59     // interrupt enable bit, and the SPI interrupt has higher priority, it needs to be enabled.
60     sei();
61
62     // run the button process algorithm every second tick
63     if (blocker++ >= 1) {
64         blocker = 0;
65     } else {
66
67         // ================================================================
68         //   B U T T O N    D E B O U N C E
69         // ================================================================
70
71         // load current button states to shift registers
72         negativePulse(IN_LOAD);
73
74         // read in button states to debounce registers
75         do {
76             debounce[p] = (debounce[p] << 1) | readValue(
       IN_1); p++;
77             debounce[p] = (debounce[p] << 1) | readValue(
       IN_2); p++;
78             debounce[p] = (debounce[p] << 1) | readValue(
       IN_3); p++;
79             debounce[p] = (debounce[p] << 1) | readValue(
       IN_4); p++;
80             pulse(IO_CLK);
81         } while (i--);
82
83         // adjust p pointer, that addresses the debounce shift registers
84         p--;
85
86         // run debounce algorithm
87         do {
88             if ((debounce[p] & DEBOUNCE_MASK) == 0) {
89
90                 // PREV=1 ==> falling edge
91                 if ((button_status[p] & PREVIOUS_MASK) != 0) {
92
93                     ////////////////////////////////////////////////
94                     spi_add_up(p);
95                     ////////////////////////////////////////////////
96
97                     // clear actual and previous states
98                     button_status[p] &= PREVIOUS_CLEAR &
       ACTUAL_CLEAR;
99
100                     // counter is less than the threshold, and no short or long press was administrated
101                     if (((button_status[p] & COUNTER_MASK) <
       COUNTER_THRESHOLD) &
102                         ((button_status[p] & SHORT_MASK) == 0) &
103                         ((button_status[p] & LONG_MASK) == 0) ) {
104
105                         ////////////////////////////////////////////////
106                         spi_add_short_press(p);
107                         ////////////////////////////////////////////////
108                     }
109
110                     // clear counter & clear lock
111                     button_status[p] &= COUNTER_CLEAR &
```

```
    LOCK_CLEAR;
112
113                     // some of the buttons/switches were released/turned off, refresh the dip status
114                     dip_update = 1;
115
116                 }
117                 continue;
118             }
119
120             if ((debounce[p] & DEBOUNCE_MASK) == DEBOUNCE_MASK ) {
121
122                 // PREV=0 ==> rising edge
123                 if ((button_status[p] & PREVIOUS_MASK) == 0) {
124
125                     ////////////////////////////////////////////////
126                     spi_add_down(p);
127                     ////////////////////////////////////////////////
128
129                     // clear actual and previous states
130                     button_status[p] |= PREVIOUS_SET |
    ACTUAL_SET;
131                     continue;
132                 }
133
134                 // no long or short press, and long press isn't locked
135                 if (((button_status[p] & LONG_MASK) == 0) &
136                     ((button_status[p] & SHORT_MASK) == 0) &
137                     ((button_status[p] & LOCK_MASK) == 0)) {
138
139                     // increment the counter
140                     button_status[p]++;
141
142                     if ((button_status[p] & COUNTER_MASK) >=
    COUNTER_THRESHOLD) {
143                         // threshold reached = long press :: lock
144                         button_status[p] |= LOCK_SET;
145
146                         // some of the buttons/switches were hold down longer than the long press threshold
147                         // refresh the dip status
148                         dip_update = 1;
149
150                         ////////////////////////////////////////////////
151                         spi_add_long_press(p);
152                         ////////////////////////////////////////////////
153                     }
154                 }
155             }
156         } while (p--);
157
158     }
159
160     // ================================================================
161     //   E N C O D E R    D E B O U N C E
162     // ================================================================
163
164     // read in values
165     encoder_debounce[0] = (encoder_debounce[0] << 1) |
    readValue(E1_A);
166     encoder_debounce[1] = (encoder_debounce[1] << 1) |
    readValue(E1_B);
167
168     encoder_debounce[2] = (encoder_debounce[2] << 1) |
    readValue(E2_A);
169     encoder_debounce[3] = (encoder_debounce[3] << 1) |
    readValue(E2_B);
170
171     encoder_debounce[4] = (encoder_debounce[4] << 1) |
    readValue(E3_A);
172     encoder_debounce[5] = (encoder_debounce[5] << 1) |
    readValue(E3_B);
173
174     // process edges
175     i = 5;
176
177     do {
178         // low level reached
179         if ((encoder_debounce[i] & E_DEBOUNCE_MASK) == 0) {
180             // store actual value
181             uint8_t temp = encoder_status[i] & E_ACTUAL_MASK;
182             // clear previous and actual value
183             encoder_status[i] &= E_PREVIOUS_CLEAR &
    E_ACTUAL_CLEAR; // + clearing the actual bit
184             // store actual value to previous
185             encoder_status[i] |= temp<<1;
186         }
187
188         // high level reached
```

```
189         if ((encoder_debounce[i] & E_DEBOUNCE_MASK) ==
    E_DEBOUNCE_MASK) {
190             // store actual value
191             uint8_t temp = encoder_status[i] & E_ACTUAL_MASK;
192             // clear previous and actual value
193             encoder_status[i] &= E_PREVIOUS_CLEAR;  // not clearing the
    actual bit, because we will or it with one
194             // store actual value to previous
195             encoder_status[i] |= (temp<<1) | E_ACTUAL_SET;  // setting the actual
    bit to one
196         }
197
198         // if current pin is master, process the edges
199         if ((encoder_status[i] & E_MASTER_MASK) != 0) {
200
201             // Previous=1 and Actual=0 ==> falling edge
202             if ( (encoder_status[i] & (E_ACTUAL_MASK |
    E_PREVIOUS_MASK) ) == 0x40 ) {
203
204                 if ((encoder_status[i+1] & E_ACTUAL_MASK) == 0) {
205                     // turn left
206                     encoder_counter[i>>1]++;
207                 } else {
208                     // turn right
209                     encoder_counter[i>>1]--;
210                 }
211
212                 ////////////////////////////////////////////
213                 spi_add_encoder(i);
214                 ////////////////////////////////////////////
215
216             }
217         }
218
219     } while (i--);
220
221
222     // ============================================================
223     //   D I P   U P D A T E
224     // ============================================================
225
226     // if something happened with the inputs, refresh the dip switches status
227     if (dip_update) {
228         cli();
229         // clear the status
230         dip_status = 0;
231         // load the actual states of the dip switches
232         // there will be no bouncing effect, because this code only runs when the switches
233         // are released (no spring action) or when they held down until a long press occurs
234         dip_status |= (button_status[16] & ACTUAL_MASK) >> 7;
235         dip_status |= (button_status[20] & ACTUAL_MASK) >> 8;
236         dip_status |= (button_status[24] & ACTUAL_MASK) >> 9;
237         dip_status |= (button_status[28] & ACTUAL_MASK) >> 10;
238         dip_status |= (button_status[0] &  ACTUAL_MASK) >> 11;
239         dip_status |= (button_status[4] &  ACTUAL_MASK) >> 12;
240         dip_status |= (button_status[8] &  ACTUAL_MASK) >> 13;
241         dip_status |= (button_status[12] & ACTUAL_MASK) >> 14;
242         sei();
243     }
244 }
```

5.6 Buttons

All the buttons of the DSP controller are driven by shift registers. These shift registers are asked periodically for new data. In every period new data runs through a debouncer algorithm that generates events, and these events are stored in a buffer, from which the spi communication program reads them out and sends to the HOST system.

Macros

- #define DEBOUNCE_MASK 0x0f
- #define PREVIOUS_MASK 0x8000
- #define PREVIOUS_SET PREVIOUS_MASK
- #define PREVIOUS_CLEAR 0x7fff
- #define ACTUAL_MASK 0x4000
- #define ACTUAL_SET ACTUAL_MASK
- #define ACTUAL_CLEAR 0xbfff
- #define SHORT_MASK 0x2000
- #define SHORT_SET SHORT_MASK
- #define SHORT_CLEAR 0xdfff
- #define LONG_MASK 0x1000
- #define LONG_SET LONG_MASK
- #define LONG_CLEAR 0xefff
- #define LOCK_MASK 0x0800
- #define LOCK_SET LOCK_MASK
- #define LOCK_CLEAR 0xf7ff
- #define COUNTER_MASK 0x07ff
- #define COUNTER_CLEAR 0xf800
- #define COUNTER_THRESHOLD 350

Functions

- uint8_t get_button_event (uint8_t p)

Variables

- volatile uint8_t debounce [32]
- volatile uint16_t button_status [32]

5.6.1 Detailed Description

All the buttons of the DSP controller are driven by shift registers. These shift registers are asked periodically for new data. In every period new data runs through a debouncer algorithm that generates events, and these events are stored in a buffer, from which the spi communication program reads them out and sends to the HOST system. The system that handles the buttons connected to the shift registers is very simple and reliable. It uses only two registers for each 32 buttons: the debounce register and the status register. The algorithm runs every 1 ms. It was triggered by a timer interrupt that kicks in every 0.5 ms, so a simple waiting code blocks it, and let run in every second interrupt.

The status register contains the following bits:

15	14	13	12	11	10 : 0
previous state	actual state	short press	long press	long lock	long press counter

- **previous state:** stores the previous debounced state of the given button

- **actual state:** contains the actual debounced state of the given button
- **short press:** it latches when a short press event is occured
- **long press:** it latches when a long press event is occured. It is in exclusive relation with the short press bit.
- **long lock:** because the debounce algorithm fires the long press event during the key is pressed down, there need to be some lock that prevents the re-triggering of the long press event when the software reads out and clears the long press flag.
- **long press** counter: counter that counts when the given button is pressed down. It will be cleared when an event is occured.

It generates three type of events:

- **actual value:** nothing fancy, the actual debounced button state
- **short press:** the user pressed and released the button before the counter reached the long press threshold
- **long press:** the user pressed and held down the button. Long press event is triggered, when the counter reaches the treshold value.

5.6.2 Macro Definition Documentation

5.6.2.1 #define ACTUAL_CLEAR 0xbfff

Variable that clears the *actual bit*. [it needs AND logic]

Definition at line 55 of file input.h.

5.6.2.2 #define ACTUAL_MASK 0x4000

Mask for the *actual bit*.

Definition at line 53 of file input.h.

5.6.2.3 #define ACTUAL_SET ACTUAL_MASK

Variable that sets the *actual bit*. For the more readable code. [it needs OR logic]

Definition at line 54 of file input.h.

5.6.2.4 #define COUNTER_CLEAR 0xf800

Variable that clears the *counter*. [it needs AND logic]

Definition at line 70 of file input.h.

5.6.2.5 #define COUNTER_MASK 0x07ff

Mask for the long press *counter*.

Definition at line 69 of file input.h.

5.6.2.6 #define COUNTER_THRESHOLD 350

The limit that determines the length of the long press event. It may be adjustable in a later release.

Definition at line 72 of file input.h.

5.6.2.7 #define DEBOUNCE_MASK 0x0f

Mask that determines the length of the debounce register. Masked out bits are ignored.

Definition at line 47 of file input.h.

5.6.2.8 #define LOCK_CLEAR 0xf7ff

Variable that clears the *lock bit*. [it needs AND logic]

Definition at line 67 of file input.h.

5.6.2.9 #define LOCK_MASK 0x0800

Mask for the *lock bit*.

Definition at line 65 of file input.h.

5.6.2.10 #define LOCK_SET LOCK_MASK

Variable that sets the *lock bit*. For the more readable code. [it needs OR logic]

Definition at line 66 of file input.h.

5.6.2.11 #define LONG_CLEAR 0xefff

Variable that clears the *long bit*. [it needs AND logic]

Definition at line 63 of file input.h.

5.6.2.12 #define LONG_MASK 0x1000

Mask for the *long bit*.

Definition at line 61 of file input.h.

5.6.2.13 #define LONG_SET LONG_MASK

Variable that sets the *long bit*. For the more readable code. [it needs OR logic]

Definition at line 62 of file input.h.

5.6.2.14 #define PREVIOUS_CLEAR 0x7fff

Variable that clears the *previous bit*. [it needs AND logic]

Definition at line 51 of file input.h.

5.6.2.15 #define PREVIOUS_MASK 0x8000

Mask for the *previous bit*.

Definition at line 49 of file input.h.

5.6.2.16 #define PREVIOUS_SET PREVIOUS_MASK

Variable that sets the *previous bit*. For the more readable code. [it needs OR logic]

Definition at line 50 of file input.h.

5.6.2.17 #define SHORT_CLEAR 0xdfff

Variable that clears the *short bit*. [it needs AND logic]

Definition at line 59 of file input.h.

5.6.2.18 #define SHORT_MASK 0x2000

Mask for the *short bit*.

Definition at line 57 of file input.h.

5.6.2.19 #define SHORT_SET SHORT_MASK

Variable that sets the *short bit*. For the more readable code. [it needs OR logic]

Definition at line 58 of file input.h.

5.6.3 Function Documentation

5.6.3.1 uint8_t get_button_event (uint8_t *p*)

Function that queries the events. If an event is triggered it returns the event, and clears the corresponding status atomically.

Parameters

in	*p*	Index to the button you want to query. [0..31]

Returns

The triggered event.

5.6.4 Variable Documentation

5.6.4.1 volatile uint16_t button_status[32]

Button's status

15	14	13	12	11	10 : 0
previous state	actual state	short press	long press	long lock	long press counter

Global array of variables that store the buttons' status information.

Definition at line 25 of file main.h.

5.6.4.2 volatile uint8_t debounce[32]

Debounce register that is functioning as a shift register. New sampled data are shifted in this register. The algorithm processes these data.

Global array of variables that act as a debounce registers fr the buttons.

Definition at line 24 of file main.h.

5.7 Encoders

The operation of the encoders is very similar to the buttons. But instead of using shift registers, it's actual value are readed out directly with some IO pins of the AVR microcontroller. These values processed through the debouncer algorithm, and than based on the result, the corresponding registers that counts the encoder's rotation are updated.

Macros

- #define E_DEBOUNCE_MASK 0x07
- #define E_PREVIOUS_MASK 0x80
- #define E_PREVIOUS_SET E_PREVIOUS_MASK
- #define E_PREVIOUS_CLEAR 0x7f
- #define E_ACTUAL_MASK 0x40
- #define E_ACTUAL_SET E_ACTUAL_MASK
- #define E_ACTUAL_CLEAR 0xbf
- #define E_MASTER_MASK 0x20
- #define E_MASTER_SET E_MASTER_MASK
- #define E_MASTER_CLEAR 0xdf

Functions

- int8_t get_encoder_value (uint8_t p)

Variables

- volatile uint8_t encoder_debounce [6]
- volatile int8_t encoder_counter [3]
- volatile uint8_t encoder_status [6]
- volatile uint8_t dip_status

5.7.1 Detailed Description

The operation of the encoders is very similar to the buttons. But instead of using shift registers, it's actual value are readed out directly with some IO pins of the AVR microcontroller. These values processed through the debouncer algorithm, and than based on the result, the corresponding registers that counts the encoder's rotation are updated. The system that do the processing of the encoders is triggered by a timer interrupt. The interrupt kicks in every 0.5 ms. It uses 3 register:

- **encoder_debounce**[6]: 8 bit debounce register that is used like a shift register
- **encoder_counter**[3]: 8 bit signed register, that stores the 3 encoders value
- **encoder_status**[6]: 8 bit status register for each encoder signal (A and B for each encoder)

Status register bits:

7	6	5	4 : 0
previous state	actual state	master	reserved

- **previous state:** it stores the previous debounced signal state
- **actual state:** it stores the actual debounced signal state
- **master:** it is set when the corresponding signal is the A encoder signal. The algorithm uses this bit to determine at which signal needs to detect the rising edge.

The algorithm uses one encoder signal edge to determine rotation direction.

5.7.2 Macro Definition Documentation

5.7.2.1 #define E_ACTUAL_CLEAR 0xbf

Variable that clears the *actual bit*. [it needs AND logic]

Definition at line 138 of file input.h.

5.7.2.2 #define E_ACTUAL_MASK 0x40

Mask for the *actual bit*.

Definition at line 136 of file input.h.

5.7.2.3 #define E_ACTUAL_SET E_ACTUAL_MASK

Variable that sets the *actual bit*. For the more readable code. [it needs OR logic]

Definition at line 137 of file input.h.

5.7.2.4 #define E_DEBOUNCE_MASK 0x07

Mask for the long press *counter*.

Definition at line 130 of file input.h.

5.7.2.5 #define E_MASTER_CLEAR 0xdf

Variable that clears the *master bit*. [it needs AND logic]

Definition at line 142 of file input.h.

5.7.2.6 #define E_MASTER_MASK 0x20

Mask for the *master bit*.

Definition at line 140 of file input.h.

5.7.2.7 #define E_MASTER_SET E_MASTER_MASK

Variable that sets the *master bit*. For the more readable code. [it needs OR logic]

Definition at line 141 of file input.h.

5.7.2.8 #define E_PREVIOUS_CLEAR 0x7f

Variable that clears the *previous bit*. [it needs AND logic]

Definition at line 134 of file input.h.

5.7.2.9 #define E_PREVIOUS_MASK 0x80

Mask for the *previous bit*.

Definition at line 132 of file input.h.

5.7.2.10 #define E_PREVIOUS_SET E_PREVIOUS_MASK

Variable that sets the *previous bit*. For the more readable code. [it needs OR logic]

Definition at line 133 of file input.h.

5.7.3 Function Documentation

5.7.3.1 int8_t get_encoder_value (uint8_t *p*)

Function that returns the current counter value of the given encoder. If the value isn't zero, it clears the counter atomically.

Parameters

in	*p*	Index to the encoder you want to query. [0..2]

Returns

The triggered event.

Definition at line 250 of file input.c.

```
250                                     {
251
252     // read out, and delete the counter value atomically
253     if (encoder_counter[p] != 0) {
254         int8_t ret = encoder_counter[p];
255         cli();
256         encoder_counter[p] = 0;
257         sei();
258         return ret;
259     }
260     return 0;
261 }
```

5.7.4 Variable Documentation

5.7.4.1 volatile uint8_t dip_status

Status of the DIP switches. 8 bit, 8 switches.

Global variable that contains the current dip status.

Definition at line 31 of file main.h.

5.7.4.2 volatile int8_t encoder_counter[3]

Counter register for each encoders. It is a signed register, it uses two's complement.

Global array that holds the actual encoder increments.

Definition at line 28 of file main.h.

5.7.4.3 volatile uint8_t encoder_debounce[6]

Debounce register that debounces the encoders raw signals.

Global array of variables that act as a debounce registers for the encoders.

Definition at line 27 of file main.h.

5.7.4.4 volatile uint8_t encoder_status[6]

Encoder's status

7	6	5	4 : 0
previous state	actual state	master	reserved

Global array that holds the encoders' status information.

Definition at line 29 of file main.h.

5.8 Outputs

All of the output interface modules that drives the interface, including the LCD display and the LED bars.

Modules

- LCD Display

  16x2 LCD display module with blue backlight.
- LED bars

  Two general purpose led bars with 8-8 leds, mainly used for volume meter.

Files

- file shift.c
- file shift.h

Functions

- void shiftOutMsbFirst (uint8_t data)
- void shiftOutLsbFirst (uint8_t data)

5.8.1 Detailed Description

All of the output interface modules that drives the interface, including the LCD display and the LED bars.

5.8.2 Function Documentation

5.8.2.1 void shiftOutLsbFirst (uint8_t *data*)

Shifts out a byte through the default output pin with the LSB bit first.

Parameters

in	*data*	The byte you want to shift out.

Definition at line 23 of file shift.c.

```
23                                      {
24     uint8_t i = 7;
25     uint8_t s = 0;
26     do {
27         ((data>>s)&0x01) ? setHigh(OUT) : setLow(OUT);
28         pulse(IO_CLK);
29         s++;
30     } while (i--);
31 }
```

5.8.2.2 void shiftOutMsbFirst (uint8_t *data*)

Shifts out a byte through the default output pin with the MSB bit first.

Parameters

in	*data*	The byte you want to shift out.

Definition at line 13 of file shift.c.

```
13                                      {
14     uint8_t i = 7;
15     uint8_t s = 0;
16     do {
17          ((data<<s)&0x80) ? setHigh(OUT) : setLow(OUT);
18          pulse(IO_CLK);
19          s++;
20     } while (i--);
21 }
```

5.9 LCD Display

16x2 LCD display module with blue backlight.

Files

- file output.c
- file output.h

Macros

- #define LCD_E 0x04
- #define LCD_BL 0x80
- #define LCD_RS 0x02

Functions

- void lcd_init ()
- void lcd_clear ()
- void lcd_home ()
- void lcd_newLine ()
- void lcd_writeString (char *s)
- void lcd_command (uint8_t value)
- void lcd_write (char value)
- void lcd_write4bits (uint8_t value, uint8_t mode)
- void refreshLeds ()

5.9.1 Detailed Description

16x2 LCD display module with blue backlight.

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

5.9.2 Macro Definition Documentation

5.9.2.1 #define LCD_BL 0x80

Backlight mask.

Definition at line 19 of file output.h.

5.9.2.2 #define LCD_E 0x04

Enable mask, that enables the data sent to the display.

Definition at line 18 of file output.h.

5.9.2.3 #define LCD_RS 0x02

Register Select mask. This bit selects between data or command.

Definition at line 20 of file output.h.

5.9.3 Function Documentation

5.9.3.1 void lcd_clear ( )

Clears the LCD display. It takes 1.64ms to clear the entire display, therefore it recommended to use it only at initialization. A better and much faster method to set the cursor back to home, and rewrite the screen.

Definition at line 56 of file output.c.

```
56                  {
57     lcd_command(0x01);
58     _delay_us(1640);
59 }
```

5.9.3.2 void lcd_command (uint8_t *value*)

Higher level communication function. It sends a command to the LCD display. ATOMIC FUNCTION

Parameters

in	*value*	The command you want to send.

Definition at line 88 of file output.c.

```
88                                  {
89     cli();
90     lcd_write4bits(value>>4, 0);
91     lcd_write4bits(value, 0);
92     sei();
93 }
```

5.9.3.3 void lcd_home ( )

Returns the cursor to the home position (upper row, first character). Very fast method. Only takes the time to transfer the data through the shift registers.

Definition at line 61 of file output.c.

```
61                  {
62     lcd_command(0x02);
63 }
```

5.9.3.4 void lcd_init ( )

Initializes the LCD display.

Definition at line 18 of file output.c.

```
18                  {
19     // we start in 8bit mode, try to set 4 bit mode
20     lcd_write4bits(0x03, 0);
21     _delay_us(4500); // wait min 4.1ms
22
23     // second try
24     lcd_write4bits(0x03, 0);
25     _delay_us(4500); // wait min 4.1ms
```

```
26
27      // third go!
28      lcd_write4bits(0x03, 0);
29      _delay_us(150);
30
31      // finally, set to 4-bit interface
32      lcd_write4bits(0x02, 0);
33
34      // we are in 4 bit mode
35      // enter entry mode: increment, no display shift
36      lcd_command(0x06);
37
38      // turn on the display with blinking display
39      lcd_command(0x0c);
40
41      // set 4bit mode again, 2 lines, 5x7 characters
42      lcd_command(0x2c);
43
44      // clear display
45      lcd_command(0x01);
46
47      // wait for clearing
48      _delay_ms(2);
49  }
```

5.9.3.5 void lcd_newLine ( )

Sets the cursor to the second line first character.

Definition at line 65 of file output.c.

```
65                      {
66      lcd_command(0xa8);
67  }
```

5.9.3.6 void lcd_write (char *value*)

Higher level communication function. It sends a character to the LCD display. ATOMIC FUNCTION

Parameters

in	*value*	The character you want to send.

Definition at line 95 of file output.c.

```
95                              {
96      cli();
97      lcd_write4bits(value>>4, 1);
98      lcd_write4bits(value, 1);
99      sei();
100 }
```

5.9.3.7 void lcd_write4bits (uint8_t *value*, uint8_t *mode*)

Low level communication function, that sends data to the LCD display through 4-wire protocol. Due to the construction of the hardware this function breaks the encapsulation and handles the led bars via the two global variables *_led_r* and *_led_l* because their shift registers are in series with the LCD's one.

Parameters

in	*value*	The command you want to send.
in	*mode*	Decides command or data will be transmitted. mode=1: data, mode=0: command.

Definition at line 103 of file output.c.

```
103                                                 {
104     int mask = (mode) ? (LCD_BL | LCD_RS) : (LCD_BL);
105     int data = 0;
106
107     data = value<<3 & 0b01111000;
108     data |= mask;
109
110     shiftOutMsbFirst(data | LCD_E);
111     shiftOutMsbFirst(_led_r);
112     shiftOutLsbFirst(_led_l);
113     pulse(OUT_LATCH);
114
115     shiftOutMsbFirst(data);
116     shiftOutMsbFirst(_led_r);
117     shiftOutLsbFirst(_led_l);
118     pulse(OUT_LATCH);
119 }
```

5.9.3.8 void lcd_writeString (char * s)

Writes a given string to the screen at the current cursor position.

Parameters

in	*s*	The string you want to print to the display.

Definition at line 74 of file output.c.

```
74                                {
75     uint8_t len = strlen(s);
76     uint8_t i;
77     for (i = 0; i < len ; i++) {
78         lcd_write(s[i]);
79         _delay_us(100);
80     }
81 }
```

5.9.3.9 void refreshLeds ( )

Low level function that refreshes the two led bar with the current values of the global led variables. It can be handy if there is no new display data to shift in but the led values were overridden. ATOMIC FUNCTION

Definition at line 122 of file output.c.

```
122                     {
123     cli();
124     shiftOutMsbFirst(LCD_BL);
125     shiftOutMsbFirst(_led_r);
126     shiftOutLsbFirst(_led_l);
127     pulse(OUT_LATCH);
128     sei();
129 }
```

5.10 LED bars

Two general purpose led bars with 8-8 leds, mainly used for volume meter.

Variables

- volatile uint8_t _led_l
- volatile uint8_t _led_r

5.10.1 Detailed Description

Two general purpose led bars with 8-8 leds, mainly used for volume meter. Global variables.

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

5.10.2 Variable Documentation

5.10.2.1 volatile uint8_t _led_l

Global variable that holds the value of the left led bar.

Global variable that holds the left led bar's current value.

Definition at line 20 of file main.h.

5.10.2.2 volatile uint8_t _led_r

Global variable that holds the value of the right led bar.

Global variable that holds the right led bar's current value..

Definition at line 21 of file main.h.

5.11 Communication

The communication interface that connects the DSP to the DSP Controller.

Modules

- SPI

  The main and only communication module to the HOST card.

5.11.1 Detailed Description

The communication interface that connects the DSP to the DSP Controller.

5.12 SPI

The main and only communication module to the HOST card.

Files

- file spi.c
- file spi.h

Macros

- #define SPI_GET_SIMPLE 0x10
- #define SPI_GET_WITH_LED 0x11
- #define SPI_GET_WITH_LCD_TOP 0x12
- #define SPI_GET_WITH_LCD_BOTTOM 0x13
- #define SPI_GET_DIP_STATUS 0x14
- #define SPI_STATE_IDLE 0
- #define SPI_STATE_TRANSMIT_SIMPLE 1
- #define SPI_STATE_TRANSMIT_LED 2
- #define SPI_STATE_TRANSMIT_LCD_TOP 3
- #define SPI_STATE_TRANSMIT_LCD_BOTTOM 4
- #define SPI_STATE_TRANSMIT_DIP_STATUS 5
- #define SPI_FLAG_NONE 0
- #define SPI_FLAG_LED 1
- #define SPI_FLAG_LCD_TOP 2
- #define SPI_FLAG_LCD_BOTTOM 3
- #define EVENT_DOWN 0x00
- #define EVENT_UP 0x40
- #define EVENT_SHORT 0x80
- #define EVENT_LONG 0xc0
- #define EVENT_TYPE_NUMPAD 0x00
- #define EVENT_TYPE_FUNCTION 0x10
- #define EVENT_TYPE_ENCODER 0x20
- #define EVENT_TYPE_DIP 0x30

Functions

- void spi_init ()
- void spi_change_transmit_buffers ()
- ISR (SPI_STC_vect)
- void spi_add_down (uint8_t id)
- void spi_add_up (uint8_t id)
- void spi_add_short_press (uint8_t id)
- void spi_add_long_press (uint8_t id)
- void spi_add_encoder (uint8_t id)

Variables

- volatile uint8_t spi_state
- volatile uint8_t spi_transmit_A_not_B
- volatile uint8_t spi_transmit_pointer_A
- volatile uint8_t spi_transmit_pointer_B
- volatile uint8_t spi_transmit_buffer_A [40]
- volatile uint8_t spi_transmit_buffer_B [40]
- volatile uint8_t * spi_transmit_pointer_READ
- volatile uint8_t * spi_transmit_buffer_READ
- volatile uint8_t * spi_transmit_pointer_WRITE
- volatile uint8_t * spi_transmit_buffer_WRITE
- volatile uint8_t spi_receive_pointer
- volatile uint8_t spi_receive_buffer [36]
- volatile uint8_t spi_flag

5.12.1 Detailed Description

The main and only communication module to the HOST card. The SPI Module is taking care of the communication with the HOST card. The host card can access the DSPController and read out the events happened between two readouts, while sending LED or LCD data. The SPI module uses a double buffered transmit mechanism to maintain error free operation and prevent event losses. It uses a very efficient and fast, custom designed protocol over the SPI communication protocol.

5.12.2 Macro Definition Documentation

5.12.2.1 #define EVENT_DOWN 0x00

Event part, that represents a down event. The SPI module will assemble the message with this parts.

Definition at line 34 of file spi.h.

5.12.2.2 #define EVENT_LONG 0xc0

Event part, that represents a long down event. The SPI module will assemble the message with this parts.

Definition at line 37 of file spi.h.

5.12.2.3 #define EVENT_SHORT 0x80

Event part, that represents a short down event. The SPI module will assemble the message with this parts.

Definition at line 36 of file spi.h.

5.12.2.4 #define EVENT_TYPE_DIP 0x30

Event type, that represents a dip event.

Definition at line 42 of file spi.h.

5.12.2.5 #define EVENT_TYPE_ENCODER 0x20

Event type, that represents a encoder event.

Definition at line 41 of file spi.h.

5.12.2.6 #define EVENT_TYPE_FUNCTION 0x10

Event type, that represents a function button event.

Definition at line 40 of file spi.h.

5.12.2.7 #define EVENT_TYPE_NUMPAD 0x00

Event type, that represents a numpad event.

Definition at line 39 of file spi.h.

5.12.2.8 #define EVENT_UP 0x40

Event part, that represents a up event. The SPI module will assemble the message with this parts.

Definition at line 35 of file spi.h.

5.12.2.9 #define SPI_FLAG_LCD_BOTTOM 3

Flag, which the SPI module signals the main loop, that there is bottom LCD data in the buffer.

Definition at line 32 of file spi.h.

5.12.2.10 #define SPI_FLAG_LCD_TOP 2

Flag, which the SPI module signals the main loop, that there is top LCD data in the buffer.

Definition at line 31 of file spi.h.

5.12.2.11 #define SPI_FLAG_LED 1

Flag, which the SPI module signals the main loop, that there is LED data in the buffer.

Definition at line 30 of file spi.h.

5.12.2.12 #define SPI_FLAG_NONE 0

Flag, which the SPI module signals the main loop, that there is no incoming data in the buffer.

Definition at line 29 of file spi.h.

5.12.2.13 #define SPI_GET_DIP_STATUS 0x14

SPI message code that is identical for the DSPController and the HOST. This message initiates a dip state readout.

Definition at line 20 of file spi.h.

5.12.2.14 #define SPI_GET_SIMPLE 0x10

SPI message code that is identical for the DSPController and the HOST. This message initiates a simple event readout.

Definition at line 16 of file spi.h.

5.12.2.15 #define SPI_GET_WITH_LCD_BOTTOM 0x13

SPI message code that is identical for the DSPController and the HOST. This message initiates an event readout with bottom lcd data, sent parallel during the readout.

Definition at line 19 of file spi.h.

5.12.2.16 #define SPI_GET_WITH_LCD_TOP 0x12

SPI message code that is identical for the DSPController and the HOST. This message initiates an event readout with top lcd data, sent parallel during the readout.

Definition at line 18 of file spi.h.

5.12.2.17 #define SPI_GET_WITH_LED 0x11

SPI message code that is identical for the DSPController and the HOST. This message initiates an event readout with led data, sent parallel during the readout.

Definition at line 17 of file spi.h.

5.12.2.18 #define SPI_STATE_IDLE 0

SPI state that represents the idle state. The SPI module is waiting for the trigger signal between readouts.

Definition at line 22 of file spi.h.

5.12.2.19 #define SPI_STATE_TRANSMIT_DIP_STATUS 5

SPI state that represents the dip status readout. The HOST reads out the dip status.

Definition at line 27 of file spi.h.

5.12.2.20 #define SPI_STATE_TRANSMIT_LCD_BOTTOM 4

SPI state that represents the simple readout with bottom LCD data. The HOST reads out the events while it sends bottom LCD data.

Definition at line 26 of file spi.h.

5.12.2.21 #define SPI_STATE_TRANSMIT_LCD_TOP 3

SPI state that represents the simple readout with top LCD data. The HOST reads out the events while it sends top LCD data.

Definition at line 25 of file spi.h.

5.12.2.22 #define SPI_STATE_TRANSMIT_LED 2

SPI state that represents the simple readout with LED data. The HOST reads out the events while it sends LED data.

Definition at line 24 of file spi.h.

5.12.2.23 #define SPI_STATE_TRANSMIT_SIMPLE 1

SPI state that represents the simple readout. The HOST reads out the events.

Definition at line 23 of file spi.h.

5.12.3 Function Documentation

5.12.3.1 ISR (SPI_STC_vect)

SPI Interrupt handler. The whole SPI state machine takes place here. It is triggered by the SPI hardware after a byte is received.

Definition at line 66 of file spi.c.

```
66                  {
67
68      //=============================================================================================
69      //  S P I   I D L E   S T A T E
70      //=============================================================================================
71      if (spi_state == SPI_STATE_IDLE) {
72
73          // Just asking for the events
74          if (SPDR == SPI_GET_SIMPLE) {
75
76              if (*spi_transmit_pointer_WRITE > 0) {
77
78                  spi_change_transmit_buffers();
79                  SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
80
81                  spi_state = SPI_STATE_TRANSMIT_SIMPLE;
82
83              }  else {
84                  SPDR = *spi_transmit_pointer_WRITE;
85              }
86
87          // Get events while sending LED data
88          } else if (SPDR == SPI_GET_WITH_LED) {
89
90              if (*spi_transmit_pointer_WRITE > 0) {
91                  spi_change_transmit_buffers();
92                  SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
93              } else {
94                  // if there is no data, it can wait after the led loop
95                  SPDR = 0;
96              }
97
98              spi_state = SPI_STATE_TRANSMIT_LED;
99              spi_receive_pointer = 2;
100
101         // Get events while sending top LCD data
102         } else if (SPDR == SPI_GET_WITH_LCD_TOP) {
103
104             if (*spi_transmit_pointer_WRITE > 0) {
105
106                 spi_change_transmit_buffers();
107                 SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
108
109             } else {
110                 SPDR = 0;
111             }
112
113             spi_state = SPI_STATE_TRANSMIT_LCD_TOP;
114             spi_receive_pointer = 16;
115
116         // Get events while sending bottom LCD data
117         } else if (SPDR == SPI_GET_WITH_LCD_BOTTOM) {
118
119             if (*spi_transmit_pointer_WRITE > 0) {
120
121                 spi_change_transmit_buffers();
122                 SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
123
124             } else {
125                 SPDR = 0;
126             }
127
128             spi_state = SPI_STATE_TRANSMIT_LCD_BOTTOM;
129             spi_receive_pointer = 16;
130
```

```
131         // Get DIP status
132         } else if (SPDR == SPI_GET_DIP_STATUS) {
133 
134             SPDR = dip_status;
135             spi_state = SPI_STATE_TRANSMIT_DIP_STATUS;
136 
137         } else {
138 
139             SPDR = *spi_transmit_pointer_WRITE;
140 
141         }
142 
143     //=============================================================================
144     //  S P I   S I M P L E   T R A N S M I T   S T A T E
145     //=============================================================================
146     } else if (spi_state == SPI_STATE_TRANSMIT_SIMPLE) {
147 
148         if (*spi_transmit_pointer_READ == 0) {
149             SPDR = *spi_transmit_pointer_WRITE;
150             spi_state = SPI_STATE_IDLE;
151         } else {
152             SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
153         }
154 
155 
156 
157     //=============================================================================
158     //  S P I   L E D   T R A N S M I T   S T A T E
159     //=============================================================================
160     } else if (spi_state == SPI_STATE_TRANSMIT_LED) {
161 
162         if (--spi_receive_pointer == 0) {
163             spi_flag = SPI_FLAG_LED;
164             spi_state = SPI_STATE_IDLE;
165             SPDR = *spi_transmit_pointer_WRITE;
166         } else {
167             if (*spi_transmit_pointer_READ > 0) {
168                 SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
169             } else {
170                 SPDR = 0;
171             }
172         }
173 
174         spi_receive_buffer[spi_receive_pointer] = SPDR;
175 
176 
177     //=============================================================================
178     //  S P I   L C D   T O P   T R A N S M I T   S T A T E
179     //=============================================================================
180     } else if (spi_state == SPI_STATE_TRANSMIT_LCD_TOP) {
181 
182         if (--spi_receive_pointer == 0) {
183             spi_flag = SPI_FLAG_LCD_TOP;
184             spi_state = SPI_STATE_IDLE;
185             SPDR = *spi_transmit_pointer_WRITE;
186         } else {
187             if (*spi_transmit_pointer_READ > 0) {
188                 SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
189             } else {
190                 SPDR = 0;
191             }
192         }
193 
194         spi_receive_buffer[spi_receive_pointer] = SPDR;
195 
196 
197     //=============================================================================
198     //  S P I   L C D   B O T T O M   T R A N S M I T   S T A T E
199     //=============================================================================
200     } else if (spi_state == SPI_STATE_TRANSMIT_LCD_BOTTOM) {
201 
202         if (--spi_receive_pointer == 0) {
203             spi_flag = SPI_FLAG_LCD_BOTTOM;
204             spi_state = SPI_STATE_IDLE;
205             SPDR = *spi_transmit_pointer_WRITE;
206         } else {
207             if (*spi_transmit_pointer_READ > 0) {
208                 SPDR = spi_transmit_buffer_READ[--(*spi_transmit_pointer_READ)];
209             } else {
210                 SPDR = 0;
211             }
212         }
213 
214         spi_receive_buffer[spi_receive_pointer] = SPDR;
215 
216 
217     //=============================================================================
```

```
218     //  S P I   D I P   S T A T U S   T R A N S M I T   S T A T E
219     //=========================================================================================
220     } else if (spi_state == SPI_STATE_TRANSMIT_DIP_STATUS) {
221
222         spi_state = SPI_STATE_IDLE;
223         SPDR = *spi_transmit_pointer_WRITE;
224
225     } // end of spi_state checkings
226
227 }
```

5.12.3.2 void spi_add_down (uint8_t *id*)

SPI module API function. This function can be called when an event occurs. This function will process the given id, and will put a down message to the actual WRITE buffer.

Parameters

in	*id*	ID that represents the event source. This is the index of the event source in the input status register.

Definition at line 235 of file spi.c.

```
235                                 {
236     cli();
237     uint8_t temp = EVENT_DOWN;
238
239     // the dip switches were excluded this code, because there is a dedicated dip switch readout mechanism
240     switch (id) {
241         case 1:     temp |= EVENT_TYPE_FUNCTION |    6; break;
242         case 2:     temp |= EVENT_TYPE_NUMPAD |        9; break;
243         case 3:     temp |= EVENT_TYPE_NUMPAD |        8; break;
244         case 5:     temp |= EVENT_TYPE_FUNCTION |    4; break;
245         case 6:     temp |= EVENT_TYPE_NUMPAD |       12; break;
246         case 7:     temp |= EVENT_TYPE_NUMPAD |        7; break;
247         case 9:     temp |= EVENT_TYPE_FUNCTION |    3; break;
248         case 10:    temp |= EVENT_TYPE_NUMPAD |       11; break;
249         case 11:    temp |= EVENT_TYPE_NUMPAD |        6; break;
250         case 13:    temp |= EVENT_TYPE_FUNCTION |    2; break;
251         case 14:    temp |= EVENT_TYPE_NUMPAD |       10; break;
252         case 15:    temp |= EVENT_TYPE_NUMPAD |        5; break;
253         case 17:    temp |= EVENT_TYPE_FUNCTION |    1; break;
254         case 18:    temp |= EVENT_TYPE_FUNCTION |    5; break;
255         case 19:    temp |= EVENT_TYPE_NUMPAD |        4; break;
256         case 21:    temp |= EVENT_TYPE_ENCODER |  3; break;
257         case 22:    temp |= EVENT_TYPE_FUNCTION |    8; break;
258         case 23:    temp |= EVENT_TYPE_NUMPAD |        3; break;
259         case 25:    temp |= EVENT_TYPE_ENCODER |  2; break;
260         case 26:    temp |= EVENT_TYPE_FUNCTION |    7; break;
261         case 27:    temp |= EVENT_TYPE_NUMPAD |        2; break;
262         case 29:    temp |= EVENT_TYPE_ENCODER |  1; break;
263         case 30:    temp |= EVENT_TYPE_FUNCTION |    9; break;
264         case 31:    temp |= EVENT_TYPE_NUMPAD |        1; break;
265         default:    temp = 0;
266     }
267
268     // if there is a real event, write it to the WRITE buffer
269     if (temp != 0) {
270         spi_transmit_buffer_WRITE[(*spi_transmit_pointer_WRITE)++] = temp;
271         if (spi_state == SPI_STATE_IDLE) {
272             SPDR = *spi_transmit_pointer_WRITE;
273         }
274     }
275
276     sei();
277 }
```

5.12.3.3 void spi_add_encoder (uint8_t *id*)

SPI module API function. This function can be called when an event occurs. This function will process the given id, and will put an encoder message to the actual WRITE buffer.

Parameters

in	*id*	ID that represents the event source. This is the index of the event source in the input status register.

Definition at line 411 of file spi.c.

```
411                                 {
412     cli();
413     uint8_t temp = 0;
414 
415     if (id == 0) {
416         temp = 13 | (encoder_counter[0]<<4);
417         encoder_counter[0] = 0;
418     } else if (id == 2) {
419         temp = 14 | (encoder_counter[1]<<4);
420         encoder_counter[1] = 0;
421     } else if (id == 4) {
422         temp = 15 | (encoder_counter[2]<<4);
423         encoder_counter[2] = 0;
424     }
425 
426     spi_transmit_buffer_WRITE[(*spi_transmit_pointer_WRITE)++] = temp;
427     if (spi_state == SPI_STATE_IDLE) {
428         SPDR = *spi_transmit_pointer_WRITE;
429     }
430 
431     sei();
432 }
```

5.12.3.4 void spi_add_long_press (uint8_t *id*)

SPI module API function. This function can be called when an event occurs. This function will process the given id, and will put a long press message to the actual WRITE buffer.

Parameters

in	*id*	ID that represents the event source. This is the index of the event source in the input status register.

Definition at line 367 of file spi.c.

```
367                                   {
368     cli();
369     uint8_t temp = EVENT_LONG;
370 
371     // the dip switches were excluded this code, because there is a dedicated dip switch readout mechanism
372     switch (id) {
373         case 1:     temp |= EVENT_TYPE_FUNCTION |    6; break;
374         case 2:     temp |= EVENT_TYPE_NUMPAD |        9; break;
375         case 3:     temp |= EVENT_TYPE_NUMPAD |        8; break;
376         case 5:     temp |= EVENT_TYPE_FUNCTION |    4; break;
377         case 6:     temp |= EVENT_TYPE_NUMPAD |       12; break;
378         case 7:     temp |= EVENT_TYPE_NUMPAD |        7; break;
379         case 9:     temp |= EVENT_TYPE_FUNCTION |    3; break;
380         case 10:    temp |= EVENT_TYPE_NUMPAD |       11; break;
381         case 11:    temp |= EVENT_TYPE_NUMPAD |        6; break;
382         case 13:    temp |= EVENT_TYPE_FUNCTION |    2; break;
383         case 14:    temp |= EVENT_TYPE_NUMPAD |       10; break;
384         case 15:    temp |= EVENT_TYPE_NUMPAD |        5; break;
385         case 17:    temp |= EVENT_TYPE_FUNCTION |    1; break;
386         case 18:    temp |= EVENT_TYPE_FUNCTION |    5; break;
387         case 19:    temp |= EVENT_TYPE_NUMPAD |        4; break;
388         case 21:    temp |= EVENT_TYPE_ENCODER |  3; break;
389         case 22:    temp |= EVENT_TYPE_FUNCTION |    8; break;
390         case 23:    temp |= EVENT_TYPE_NUMPAD |        3; break;
391         case 25:    temp |= EVENT_TYPE_ENCODER |  2; break;
392         case 26:    temp |= EVENT_TYPE_FUNCTION |    7; break;
393         case 27:    temp |= EVENT_TYPE_NUMPAD |        2; break;
394         case 29:    temp |= EVENT_TYPE_ENCODER |  1; break;
395         case 30:    temp |= EVENT_TYPE_FUNCTION |    9; break;
396         case 31:    temp |= EVENT_TYPE_NUMPAD |        1; break;
397         default:    temp = 0;
398     }
399 
400     // if there is a real event, write it to the WRITE buffer
401     if (temp != 0) {
402         spi_transmit_buffer_WRITE[(*spi_transmit_pointer_WRITE)++] = temp;
```

```
403         if (spi_state == SPI_STATE_IDLE) {
404             SPDR = *spi_transmit_pointer_WRITE;
405         }
406     }
407
408     sei();
409 }
```

5.12.3.5 void spi_add_short_press (uint8_t *id*)

SPI module API function. This function can be called when an event occurs. This function will process the given id, and will put a short press message to the actual WRITE buffer.

Parameters

in	*id*	ID that represents the event source. This is the index of the event source in the input status register.

Definition at line 323 of file spi.c.

```
323                                     {
324     cli();
325     uint8_t temp = EVENT_SHORT;
326
327     // the dip switches were excluded this code, because there is a dedicated dip switch readout mechanism
328     switch (id) {
329         case 1:     temp |= EVENT_TYPE_FUNCTION |    6; break;
330         case 2:     temp |= EVENT_TYPE_NUMPAD |      9; break;
331         case 3:     temp |= EVENT_TYPE_NUMPAD |      8; break;
332         case 5:     temp |= EVENT_TYPE_FUNCTION |    4; break;
333         case 6:     temp |= EVENT_TYPE_NUMPAD |      12; break;
334         case 7:     temp |= EVENT_TYPE_NUMPAD |      7; break;
335         case 9:     temp |= EVENT_TYPE_FUNCTION |    3; break;
336         case 10:    temp |= EVENT_TYPE_NUMPAD |      11; break;
337         case 11:    temp |= EVENT_TYPE_NUMPAD |      6; break;
338         case 13:    temp |= EVENT_TYPE_FUNCTION |    2; break;
339         case 14:    temp |= EVENT_TYPE_NUMPAD |      10; break;
340         case 15:    temp |= EVENT_TYPE_NUMPAD |      5; break;
341         case 17:    temp |= EVENT_TYPE_FUNCTION |    1; break;
342         case 18:    temp |= EVENT_TYPE_FUNCTION |    5; break;
343         case 19:    temp |= EVENT_TYPE_NUMPAD |      4; break;
344         case 21:    temp |= EVENT_TYPE_ENCODER |  3; break;
345         case 22:    temp |= EVENT_TYPE_FUNCTION |    8; break;
346         case 23:    temp |= EVENT_TYPE_NUMPAD |      3; break;
347         case 25:    temp |= EVENT_TYPE_ENCODER |  2; break;
348         case 26:    temp |= EVENT_TYPE_FUNCTION |    7; break;
349         case 27:    temp |= EVENT_TYPE_NUMPAD |      2; break;
350         case 29:    temp |= EVENT_TYPE_ENCODER |  1; break;
351         case 30:    temp |= EVENT_TYPE_FUNCTION |    9; break;
352         case 31:    temp |= EVENT_TYPE_NUMPAD |      1; break;
353         default:    temp = 0;
354     }
355
356     // if there is a real event, write it to the WRITE buffer
357     if (temp != 0) {
358         spi_transmit_buffer_WRITE[(*spi_transmit_pointer_WRITE)++] = temp;
359         if (spi_state == SPI_STATE_IDLE) {
360             SPDR = *spi_transmit_pointer_WRITE;
361         }
362     }
363
364     sei();
365 }
```

5.12.3.6 void spi_add_up (uint8_t *id*)

SPI module API function. This function can be called when an event occurs. This function will process the given id, and will put an up message to the actual WRITE buffer.

Parameters

in	*id*	ID that represents the event source. This is the index of the event source in the input status register.

Definition at line 279 of file spi.c.

```
279                             {
280     cli();
281     uint8_t temp = EVENT_UP;
282 
283     // the dip switches were excluded this code, because there is a dedicated dip switch readout mechanism
284     switch (id) {
285         case 1:     temp |= EVENT_TYPE_FUNCTION |    6; break;
286         case 2:     temp |= EVENT_TYPE_NUMPAD |      9; break;
287         case 3:     temp |= EVENT_TYPE_NUMPAD |      8; break;
288         case 5:     temp |= EVENT_TYPE_FUNCTION |    4; break;
289         case 6:     temp |= EVENT_TYPE_NUMPAD |      12; break;
290         case 7:     temp |= EVENT_TYPE_NUMPAD |      7; break;
291         case 9:     temp |= EVENT_TYPE_FUNCTION |    3; break;
292         case 10:    temp |= EVENT_TYPE_NUMPAD |      11; break;
293         case 11:    temp |= EVENT_TYPE_NUMPAD |      6; break;
294         case 13:    temp |= EVENT_TYPE_FUNCTION |    2; break;
295         case 14:    temp |= EVENT_TYPE_NUMPAD |      10; break;
296         case 15:    temp |= EVENT_TYPE_NUMPAD |      5; break;
297         case 17:    temp |= EVENT_TYPE_FUNCTION |    1; break;
298         case 18:    temp |= EVENT_TYPE_FUNCTION |    5; break;
299         case 19:    temp |= EVENT_TYPE_NUMPAD |      4; break;
300         case 21:    temp |= EVENT_TYPE_ENCODER |  3; break;
301         case 22:    temp |= EVENT_TYPE_FUNCTION |    8; break;
302         case 23:    temp |= EVENT_TYPE_NUMPAD |      3; break;
303         case 25:    temp |= EVENT_TYPE_ENCODER |  2; break;
304         case 26:    temp |= EVENT_TYPE_FUNCTION |    7; break;
305         case 27:    temp |= EVENT_TYPE_NUMPAD |      2; break;
306         case 29:    temp |= EVENT_TYPE_ENCODER |  1; break;
307         case 30:    temp |= EVENT_TYPE_FUNCTION |    9; break;
308         case 31:    temp |= EVENT_TYPE_NUMPAD |      1; break;
309         default:    temp = 0;
310     }
311 
312     // if there is a real event, write it to the WRITE buffer
313     if (temp != 0) {
314         spi_transmit_buffer_WRITE[(*spi_transmit_pointer_WRITE)++] = temp;
315         if (spi_state == SPI_STATE_IDLE) {
316             SPDR = *spi_transmit_pointer_WRITE;
317         }
318     }
319 
320     sei();
321 }
```

5.12.3.7 void spi_change_transmit_buffers ( )

This function will change the pointers between the READ and WRITE buffers. The previous READ buffer will be the actual WRITE buffer and vice versa.

Definition at line 39 of file spi.c.

```
39                                      {
40     // flip the decider variable
41     spi_transmit_A_not_B = !spi_transmit_A_not_B;
42 
43     // and change the READ buffer to the WRITE buffer and vice versa
44     if (spi_transmit_A_not_B) {
45 
46         spi_transmit_pointer_WRITE = &
      spi_transmit_pointer_A;
47         spi_transmit_buffer_WRITE =
      spi_transmit_buffer_A;
48 
49         spi_transmit_pointer_READ = &
      spi_transmit_pointer_B;
50         spi_transmit_buffer_READ = spi_transmit_buffer_B;
51 
52     } else {
53 
54         spi_transmit_pointer_WRITE = &
      spi_transmit_pointer_B;
55         spi_transmit_buffer_WRITE =
```

```
      spi_transmit_buffer_B;
56
57          spi_transmit_pointer_READ = &
      spi_transmit_pointer_A;
58          spi_transmit_buffer_READ = spi_transmit_buffer_A;
59
60      }
61  }
```

5.12.3.8 void spi_init ( )

This function will initialize the SPI module. It configures the SPI hardware as a slave, set the IO pins accordingly, initializes the SPI module's state variable, and the double buffered mechanism.

Definition at line 14 of file spi.c.

```
14                  {
15
16      // set MISO output
17      setOutput(MISO);
18
19      // the DSPController will be the slave, therefore it doesn't need to set the SPI clock speed.
20      // enable SPI and SPI interrupt
21      SPCR = (1<<SPE) | (1<<SPIE);
22
23      SPDR = 0x00;
24
25      spi_state = SPI_STATE_IDLE;
26      spi_flag = SPI_FLAG_NONE;
27
28      // prepare the double buffered transmit mechanism
29      spi_transmit_A_not_B = 1;
30      spi_transmit_pointer_A = 0;
31      spi_transmit_pointer_B = 0;
32
33      spi_transmit_pointer_WRITE = &
      spi_transmit_pointer_A;
34      spi_transmit_buffer_WRITE = spi_transmit_buffer_A;
35      spi_transmit_pointer_READ = &spi_transmit_pointer_B;
36      spi_transmit_buffer_READ = spi_transmit_buffer_B;
37  }
```

5.12.4 Variable Documentation

5.12.4.1 volatile uint8_t spi_flag

Flag variable that holds the latest incoming event type.

Global variable that .

Definition at line 49 of file main.h.

5.12.4.2 volatile uint8_t spi_receive_buffer[36]

SPI receive buffer that holds incoming LCD and LED data.

Global variable that .

Definition at line 48 of file main.h.

5.12.4.3 volatile uint8_t spi_receive_pointer

Index variable to index the last item in the incoming data buffer used for LCD and LED data.

Global variable that .

Definition at line 47 of file main.h.

5.12.4.4 volatile uint8_t spi_state

SPI state variable, that holds the state of the SPI module.

Global variable that holds the actual SPI state.

Definition at line 34 of file main.h.

5.12.4.5 volatile uint8_t spi_transmit_A_not_B

Variable that decides which SPI buffer will be the next readable buffer.

Global variable that decides which SPI buffer will be the next readable buffer.

Definition at line 36 of file main.h.

5.12.4.6 volatile uint8_t spi_transmit_buffer_A[40]

Transmit buffer A, that holds the events that occurred during the input processing.

Global variable that .

Definition at line 39 of file main.h.

5.12.4.7 volatile uint8_t spi_transmit_buffer_B[40]

Transmit buffer B, that holds the events that occurred during the input processing.

Global variable that .

Definition at line 40 of file main.h.

5.12.4.8 volatile uint8_t* spi_transmit_buffer_READ

Pointer for code simplification that points to the actual READ buffer.

Global variable that .

Definition at line 43 of file main.h.

5.12.4.9 volatile uint8_t* spi_transmit_buffer_WRITE

Pointer for code simplification that points to the actual WRITE buffer.

Global variable that .

Definition at line 45 of file main.h.

5.12.4.10 volatile uint8_t spi_transmit_pointer_A

Index variable that indexes the last item in the A buffer.

Global variable that .

Definition at line 37 of file main.h.

5.12.4.11 volatile uint8_t spi_transmit_pointer_B

Index variable that indexes the last item in the B buffer.

Global variable that .

Definition at line 38 of file main.h.

5.12.4.12 volatile uint8_t* spi_transmit_pointer_READ

Pointer for code simplification that points to the actual READ pointer that indexes the last item in the actual READ buffer.

Global variable that .

Definition at line 42 of file main.h.

5.12.4.13 volatile uint8_t* spi_transmit_pointer_WRITE

Pointer for code simplification that points to the actual WRITE pointer that indexes the last item in the actual WRITE buffer.

Global variable that .

Definition at line 44 of file main.h.

5.13 USART Logger

A simple lightweight console debug tool that redirects standard io stream to the USART hardware.

Files

- file usart_logger.c
- file usart_logger.h

Macros

- #define F_CPU 16000000UL
- #define BAUD 57600
- #define LOGGER_ON_
- #define LOG(A,...) printf(A,##__VA_ARGS__)

Functions

- int usart_putchar (char c, FILE ∗stream)
- int usart_getchar (FILE ∗stream)
- void usart_logger_init (void)

Variables

- FILE usart_output = FDEV_SETUP_STREAM(usart_putchar, NULL, _FDEV_SETUP_WRITE)
- FILE usart_input = FDEV_SETUP_STREAM(NULL, usart_getchar, _FDEV_SETUP_READ)

5.13.1 Detailed Description

A simple lightweight console debug tool that redirects standard io stream to the USART hardware.

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

This module allows the developer a convenient way to debug the code without a hardware debugger. It uses the standard io stream provided by the C library and redirects it to the AVR USART hardware.

It can be compiled to the code on demand. You can turn it on, by defining the LOGGER_ON_ symbol. The comment macro will be transparent to the compiler if the module is turned off.

Default USART configurations

USART Configuration	values
baudrate	57600
number of bits	8

parity	NO
stop bits	1

To use this module, simply write your debug messages with the LOG() macro exactly the same way as you may use the printf() function, and the message will be sent through the serial hardware. The macro ensures that the debug module stay transparent if you not use it, i.e. you comment out the LOGGER_ON_ symbol, the compiler won't compile it. It could be handy if you are working on a large project, where the flash storage is the bottleneck, or you do not need the final hardware to print diagnostic messages through it's serial port. You wouldn't need to delete every single LOG() instruction, you only need to commnet out the LOGGER_ON_ symbol, and all debug messages will be gone.

5.13.2 Macro Definition Documentation

5.13.2.1 #define BAUD 57600

UART baudrate

Definition at line 20 of file usart_logger.c.

5.13.2.2 #define F_CPU 16000000UL

CPU clock frequency

Definition at line 16 of file usart_logger.c.

5.13.2.3 #define LOG(*A, ...*) printf(A,##__VA_ARGS__)

Logger macro that wraps around the *printf* function.

Definition at line 17 of file usart_logger.h.

5.13.2.4 #define LOGGER_ON_

Comment out this preprocessor variable and USART Logger will not compile to your binary file.

Definition at line 14 of file usart_logger.h.

5.13.3 Function Documentation

5.13.3.1 int usart_getchar (FILE * *stream*)

Receiving a character via the USART interface.

Parameters

in	*stream*	Reference to the redirected standard IO stream.

Returns

The received character.

Definition at line 34 of file usart_logger.c.

```
34                                      {
35          loop_until_bit_is_set(UCSR0A, RXC0);
36          return UDR0;
37      }
```

5.13.3.2 void usart_logger_init (void)

Initialization function that configures the UART hardware and redirects the standard IO stream to the UART itself.

Definition at line 44 of file usart_logger.c.

```
44                              {
45 #ifdef LOGGER_ON_
46
47     UBRR0H = UBRRH_VALUE;
48     UBRR0L = UBRRL_VALUE;
49
50     #if USE_2X
51     UCSR0A |= _BV(U2X0);
52     #else
53     UCSR0A &= ~(_BV(U2X0));
54     #endif
55
56     UCSR0C = _BV(UCSZ01) | _BV(UCSZ00); /* 8-bit data */
57     UCSR0B = _BV(RXEN0) | _BV(TXEN0);   /* Enable RX and TX */
58
59     stdout = &usart_output;
60     stdin  = &usart_input;
61
62 #endif // LOGGER_ON_
63 }
```

5.13.3.3 int usart_putchar (char *c*, FILE * *stream*)

Sending a character via the USART interface.

Parameters

in	*c*	Character to send.
in	*stream*	Reference to the redirected standard IO stream.

Definition at line 25 of file usart_logger.c.

```
25                                              {
26         if (c == '\n') {
27             usart_putchar('\r', stream);
28         }
29         loop_until_bit_is_set(UCSR0A, UDRE0);
30         UDR0 = c;
31         return 0;
32     }
```

5.13.4 Variable Documentation

5.13.4.1 FILE usart_input = FDEV_SETUP_STREAM(NULL, usart_getchar, _FDEV_SETUP_READ)

Variable that represents the new input stream.

Definition at line 40 of file usart_logger.c.

5.13.4.2 FILE usart_output = FDEV_SETUP_STREAM(usart_putchar, NULL, _FDEV_SETUP_WRITE)

Variable that represents the new output stream.

Definition at line 39 of file usart_logger.c.

Chapter 6

File Documentation

6.1 bsp.h File Reference

```
#include "includes.h"
```

Macros

- #define IO_CLK IO_CLK
- #define OUT_LATCH OUT_LATCH
- #define IN_LOAD IN_LOAD
- #define OUT OUT
- #define IN_1 IN_1
- #define IN_2 IN_2
- #define IN_3 IN_3
- #define IN_4 IN_4
- #define E1_A E1_A
- #define E1_B E1_B
- #define E2_A E2_A
- #define E2_B E2_B
- #define E3_A E3_A
- #define E3_B E3_B
- #define MOSI MOSI
- #define MISO MISO
- #define SCK SCK
- #define SS SS
- #define MOSI_DDR DDRB
- #define MOSI_PORT PORTB
- #define MOSI_PIN PINB
- #define MOSI_NAME PB3
- #define MISO_DDR DDRB
- #define MISO_PORT PORTB
- #define MISO_PIN PINB
- #define MISO_NAME PB4
- #define SS_DDR DDRB
- #define SS_PORT PORTB
- #define SS_PIN PINB
- #define SS_NAME PB2
- #define SCK_DDR DDRB

- #define SCK_PORT PORTB
- #define SCK_PIN PINB
- #define SCK_NAME PB5
- #define IO_CLK_DDR DDRC
- #define IO_CLK_PORT PORTC
- #define IO_CLK_PIN PINC
- #define IO_CLK_NAME PC0
- #define OUT_LATCH_DDR DDRC
- #define OUT_LATCH_PORT PORTC
- #define OUT_LATCH_PIN PINC
- #define OUT_LATCH_NAME PC1
- #define IN_LOAD_DDR DDRC
- #define IN_LOAD_PORT PORTC
- #define IN_LOAD_PIN PINC
- #define IN_LOAD_NAME PC2
- #define OUT_DDR DDRC
- #define OUT_PORT PORTC
- #define OUT_PIN PINC
- #define OUT_NAME PC3
- #define IN_1_DDR DDRC
- #define IN_1_PORT PORTC
- #define IN_1_PIN PINC
- #define IN_1_NAME PC4
- #define IN_2_DDR DDRC
- #define IN_2_PORT PORTC
- #define IN_2_PIN PINC
- #define IN_2_NAME PC5
- #define IN_3_DDR DDRB
- #define IN_3_PORT PORTB
- #define IN_3_PIN PINB
- #define IN_3_NAME PB0
- #define IN_4_DDR DDRB
- #define IN_4_PORT PORTB
- #define IN_4_PIN PINB
- #define IN_4_NAME PB1
- #define E1_A_DDR DDRD
- #define E1_A_PORT PORTD
- #define E1_A_PIN PIND
- #define E1_A_NAME PD3
- #define E1_B_DDR DDRD
- #define E1_B_PORT PORTD
- #define E1_B_PIN PIND
- #define E1_B_NAME PD2
- #define E2_A_DDR DDRD
- #define E2_A_PORT PORTD
- #define E2_A_PIN PIND
- #define E2_A_NAME PD5
- #define E2_B_DDR DDRD
- #define E2_B_PORT PORTD
- #define E2_B_PIN PIND
- #define E2_B_NAME PD4
- #define E3_A_DDR DDRD
- #define E3_A_PORT PORTD
- #define E3_A_PIN PIND
- #define E3_A_NAME PD6

- #define E3_B_DDR DDRD
- #define E3_B_PORT PORTD
- #define E3_B_PIN PIND
- #define E3_B_NAME PD7
- #define IN_1_A 0x01
- #define IN_1_B 0x02
- #define IN_1_C 0x04
- #define IN_1_D 0x08
- #define IN_1_E 0x10
- #define IN_1_F 0x20
- #define IN_1_G 0x40
- #define IN_1_H 0x80
- #define IN_2_A 0x01
- #define IN_2_B 0x02
- #define IN_2_C 0x04
- #define IN_2_D 0x08
- #define IN_2_E 0x10
- #define IN_2_F 0x20
- #define IN_2_G 0x40
- #define IN_2_H 0x80
- #define IN_3_A 0x01
- #define IN_3_B 0x02
- #define IN_3_C 0x04
- #define IN_3_D 0x08
- #define IN_3_E 0x10
- #define IN_3_F 0x20
- #define IN_3_G 0x40
- #define IN_3_H 0x80
- #define IN_4_A 0x01
- #define IN_4_B 0x02
- #define IN_4_C 0x04
- #define IN_4_D 0x08
- #define IN_4_E 0x10
- #define IN_4_F 0x20
- #define IN_4_G 0x40
- #define IN_4_H 0x80
- #define DIP_1 IN_1_A
- #define DIP_2 IN_1_B
- #define DIP_3 IN_1_C
- #define DIP_4 IN_1_D
- #define DIP_5 IN_1_E
- #define DIP_6 IN_1_F
- #define DIP_7 IN_1_G
- #define DIP_8 IN_1_H
- #define E1 IN_2_A
- #define E2 IN_2_B
- #define E3 IN_2_C
- #define F1 IN_2_D
- #define F2 IN_2_E
- #define F3 IN_2_F
- #define F4 IN_2_G
- #define A1 IN_3_D
- #define A2 IN_2_H
- #define A3 IN_3_B
- #define A4 IN_3_C

- #define A5 IN_3_A
- #define N1 IN_4_A
- #define N2 IN_4_B
- #define N3 IN_4_C
- #define N4 IN_4_D
- #define N5 IN_4_E
- #define N6 IN_4_F
- #define N7 IN_4_G
- #define N8 IN_4_H
- #define N9 IN_3_H
- #define N10 IN_3_E
- #define N11 IN_3_F
- #define N12 IN_3_G
- #define output(ddr, name) ((ddr) |= (1 << (name)))
- #define input(ddr, name) ((ddr) &= ~(1 << (name)))
- #define setOutput(name) output(name##_DDR,name##_NAME)
- #define setInput(name) input(name##_DDR,name##_NAME)
- #define setInputWPullup(name)
- #define toggle(pin, name) ((pin) |= (1 << (name)))
- #define low(port, name) ((port) &= ~(1 << (name)))
- #define high(port, name) ((port) |= (1 << (name)))
- #define setLow(name) low(name##_PORT,name##_NAME)
- #define setHigh(name) high(name##_PORT,name##_NAME)
- #define read(pin, name) (((pin) & (1<<name)) >> name)
- #define readValue(name) read(name##_PIN,name##_NAME)
- #define pulse(name)
- #define negativePulse(name)
- #define setLed(L, R)

6.1.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file bsp.h.

6.2 includes.h File Reference

```
#include <avr/io.h>
#include <stdio.h>
#include <stdint.h>
#include "usart_logger.h"
#include "bsp.h"
#include "shift.h"
#include "output.h"
#include "input.h"
#include "spi.h"
#include <util/delay.h>
#include <avr/interrupt.h>
```

Macros

- #define F_CPU 16000000UL

Variables

- volatile uint8_t _led_l
- volatile uint8_t _led_r

6.2.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file includes.h.

6.3 input.c File Reference

```
#include "input.h"
```

Functions

- void input_init ()
- ISR (TIMER0_COMPA_vect)
- int8_t get_encoder_value (uint8_t p)

6.3.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file input.c.

6.4 input.h File Reference

```
#include "includes.h"
```

Macros

- #define DEBOUNCE_MASK 0x0f
- #define PREVIOUS_MASK 0x8000
- #define PREVIOUS_SET PREVIOUS_MASK
- #define PREVIOUS_CLEAR 0x7fff
- #define ACTUAL_MASK 0x4000
- #define ACTUAL_SET ACTUAL_MASK
- #define ACTUAL_CLEAR 0xbfff
- #define SHORT_MASK 0x2000
- #define SHORT_SET SHORT_MASK
- #define SHORT_CLEAR 0xdfff
- #define LONG_MASK 0x1000
- #define LONG_SET LONG_MASK
- #define LONG_CLEAR 0xefff
- #define LOCK_MASK 0x0800
- #define LOCK_SET LOCK_MASK
- #define LOCK_CLEAR 0xf7ff
- #define COUNTER_MASK 0x07ff
- #define COUNTER_CLEAR 0xf800
- #define COUNTER_THRESHOLD 350
- #define E_DEBOUNCE_MASK 0x07
- #define E_PREVIOUS_MASK 0x80
- #define E_PREVIOUS_SET E_PREVIOUS_MASK
- #define E_PREVIOUS_CLEAR 0x7f
- #define E_ACTUAL_MASK 0x40
- #define E_ACTUAL_SET E_ACTUAL_MASK
- #define E_ACTUAL_CLEAR 0xbf
- #define E_MASTER_MASK 0x20
- #define E_MASTER_SET E_MASTER_MASK
- #define E_MASTER_CLEAR 0xdf

Functions

- uint8_t get_button_event (uint8_t p)
- int8_t get_encoder_value (uint8_t p)
- void input_init ()

Variables

- volatile uint8_t debounce [32]
- volatile uint16_t button_status [32]
- volatile uint8_t encoder_debounce [6]
- volatile int8_t encoder_counter [3]
- volatile uint8_t encoder_status [6]
- volatile uint8_t dip_status

6.4.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file input.h.

6.5 main.c File Reference

```
#include "includes.h"
#include "main.h"
```

Functions

- int main (void)
- void io_init ()
- void timer_init ()

6.5.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file main.c.

6.6 main.h File Reference

Functions

- void timer_init ()
- void io_init ()
- int main (void)

Variables

- volatile uint8_t _led_l = 0
- volatile uint8_t _led_r = 0
- volatile uint8_t debounce [32]

- volatile uint16_t button_status [32]
- volatile uint8_t encoder_debounce [6]
- volatile int8_t encoder_counter [3]
- volatile uint8_t encoder_status [6]
- volatile uint8_t dip_status
- volatile uint8_t spi_state
- volatile uint8_t spi_transmit_A_not_B
- volatile uint8_t spi_transmit_pointer_A
- volatile uint8_t spi_transmit_pointer_B
- volatile uint8_t spi_transmit_buffer_A [40]
- volatile uint8_t spi_transmit_buffer_B [40]
- volatile uint8_t * spi_transmit_pointer_READ
- volatile uint8_t * spi_transmit_buffer_READ
- volatile uint8_t * spi_transmit_pointer_WRITE
- volatile uint8_t * spi_transmit_buffer_WRITE
- volatile uint8_t spi_receive_pointer
- volatile uint8_t spi_receive_buffer [36]
- volatile uint8_t spi_flag

6.6.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file main.h.

6.7 output.c File Reference

```
#include "output.h"
#include <string.h>
```

Functions

- void lcd_init ()
- void lcd_clear ()
- void lcd_home ()
- void lcd_newLine ()
- void lcd_writeString (char *s)
- void lcd_command (uint8_t value)
- void lcd_write (char value)
- void lcd_write4bits (uint8_t value, uint8_t mode)
- void refreshLeds ()

6.7.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

Definition in file output.c.

6.8 output.h File Reference

```
#include "includes.h"
```

Macros

- #define LCD_E 0x04
- #define LCD_BL 0x80
- #define LCD_RS 0x02

Functions

- void lcd_init ()
- void lcd_clear ()
- void lcd_home ()
- void lcd_newLine ()
- void lcd_writeString (char *s)
- void lcd_command (uint8_t value)
- void lcd_write (char value)
- void lcd_write4bits (uint8_t value, uint8_t mode)
- void refreshLeds ()

6.8.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file output.h.

6.9 shift.c File Reference

```
#include "shift.h"
```

Functions

- void shiftOutMsbFirst (uint8_t data)
- void shiftOutLsbFirst (uint8_t data)

6.9.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file shift.c.

6.10 shift.h File Reference

```
#include "includes.h"
```

Functions

- void shiftOutMsbFirst (uint8_t data)
- void shiftOutLsbFirst (uint8_t data)

6.10.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file shift.h.

6.11 spi.c File Reference

```
#include "includes.h"
```

Functions

- void spi_init ()
- void spi_change_transmit_buffers ()
- ISR (SPI_STC_vect)

- void spi_add_down (uint8_t id)
- void spi_add_up (uint8_t id)
- void spi_add_short_press (uint8_t id)
- void spi_add_long_press (uint8_t id)
- void spi_add_encoder (uint8_t id)

6.11.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file spi.c.

6.12 spi.h File Reference

Macros

- #define SPI_GET_SIMPLE 0x10
- #define SPI_GET_WITH_LED 0x11
- #define SPI_GET_WITH_LCD_TOP 0x12
- #define SPI_GET_WITH_LCD_BOTTOM 0x13
- #define SPI_GET_DIP_STATUS 0x14
- #define SPI_STATE_IDLE 0
- #define SPI_STATE_TRANSMIT_SIMPLE 1
- #define SPI_STATE_TRANSMIT_LED 2
- #define SPI_STATE_TRANSMIT_LCD_TOP 3
- #define SPI_STATE_TRANSMIT_LCD_BOTTOM 4
- #define SPI_STATE_TRANSMIT_DIP_STATUS 5
- #define SPI_FLAG_NONE 0
- #define SPI_FLAG_LED 1
- #define SPI_FLAG_LCD_TOP 2
- #define SPI_FLAG_LCD_BOTTOM 3
- #define EVENT_DOWN 0x00
- #define EVENT_UP 0x40
- #define EVENT_SHORT 0x80
- #define EVENT_LONG 0xc0
- #define EVENT_TYPE_NUMPAD 0x00
- #define EVENT_TYPE_FUNCTION 0x10
- #define EVENT_TYPE_ENCODER 0x20
- #define EVENT_TYPE_DIP 0x30

Functions

- void spi_add_down (uint8_t id)
- void spi_add_up (uint8_t id)
- void spi_add_short_press (uint8_t id)
- void spi_add_long_press (uint8_t id)
- void spi_add_encoder (uint8_t id)
- void spi_change_transmit_buffers ()
- void spi_init ()

Variables

- volatile uint8_t spi_state
- volatile uint8_t spi_transmit_A_not_B
- volatile uint8_t spi_transmit_pointer_A
- volatile uint8_t spi_transmit_pointer_B
- volatile uint8_t spi_transmit_buffer_A [40]
- volatile uint8_t spi_transmit_buffer_B [40]
- volatile uint8_t * spi_transmit_pointer_READ
- volatile uint8_t * spi_transmit_buffer_READ
- volatile uint8_t * spi_transmit_pointer_WRITE
- volatile uint8_t * spi_transmit_buffer_WRITE
- volatile uint8_t spi_receive_pointer
- volatile uint8_t spi_receive_buffer [36]
- volatile uint8_t spi_flag

6.12.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file spi.h.

6.13 usart_logger.c File Reference

```
#include <avr/io.h>
#include <stdio.h>
#include "usart_logger.h"
#include <util/setbaud.h>
```

Macros

- #define F_CPU 16000000UL
- #define BAUD 57600

Functions

- int usart_putchar (char c, FILE *stream)
- int usart_getchar (FILE *stream)
- void usart_logger_init (void)

Variables

- FILE usart_output = FDEV_SETUP_STREAM(usart_putchar, NULL, _FDEV_SETUP_WRITE)
- FILE usart_input = FDEV_SETUP_STREAM(NULL, usart_getchar, _FDEV_SETUP_READ)

6.13.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

Definition in file usart_logger.c.

6.14 usart_logger.h File Reference

Macros

- #define LOGGER_ON_
- #define LOG(A,...) printf(A,##__VA_ARGS__)

Functions

- int usart_putchar (char c, FILE *stream)
- int usart_getchar (FILE *stream)
- void usart_logger_init (void)

6.14.1 Detailed Description

Author

Tibor Simon tiborsimon@tibor-simon.com

Version

1.0

License

Definition in file usart_logger.h.