```
/**
 * @file
 * File:                powerspy.h
 * Author:              Manuel Federanko
 * Version:             1.0
 * Comments:
 * Revision history:
 */

#include <xc.h>
#include "types.h"

#ifndef __POWERSPY_H
#define __POWERSPY_H

#ifdef  __cplusplus
extern "C" {
#endif

#pragma config FOSC = INTOSC    // Oscillator Selection (INTOSC oscillator: I/O f
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select (MCLR/VPP pin functio
#pragma config CP = OFF         // Flash Program Memory Code Protection (Program 
#pragma config CPD = OFF        // Data Memory Code Protection (Data memory code 
#pragma config BOREN = OFF      // Brown-out Reset Enable (Brown-out Reset disabl
#pragma config CLKOUTEN = OFF   // Clock Out Enable (CLKOUT function is disabled.
#pragma config IESO = OFF       // Internal/External Switchover (Internal/Externa
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable (Fail-Safe Cloc

        // CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write prot
#pragma config PLLEN = ON       // PLL Enable (4x PLL disabled)
#pragma config STVREN = OFF     // Stack Overflow/Underflow Reset Enable (Stack O
#pragma config BORV = LO        // Brown-out Reset Voltage Selection (Brown-out R
#pragma config LVP = OFF        // Low-Voltage Programming Enable (High-voltage o

        //xc8 gives a warning when converting to lower data types
        //even when casting to the appropriate type
#pragma warning push
#pragma warning disable 752
#pragma warning disable 520
#pragma warning pop

        //print specific defines
#define _XTAL_FREQ              32000000
#define RX                      RB1
#define TX                      RB2

#define IN_FREQ                 50

#define CURRENT_VAL_IN          RB5
#define CURRENT_PHA_IN          RA0

#define VOLTAGE_VAL_IN          RB4
#define VOLTAGE_PHA_IN          RA1
```

```

#define DISPLAY_LAT             RA3
#define DISPLAY_CLK             RA4
#define DISPLAY_DATA            RA7

#define SHIFT_DIR_MSBFIRST      1
#define SHIFT_DIR_LSBFIRST      0

#define STATUS_LED              RA6
#define BUTTON                  RB3

#define PWM_OUT_GEN_VOLT        RB0
#define PWM_IN_REF              RA2

#define SHIFT_REG_LEN           7
#define SHIFT_DELAY             NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();\
                                NOP();

#define RET_OK                  0
#define RET_NOK                 1

#define K_RAWCURRENT            'C'
#define K_OFFS                  'o'
#define K_CURRENT               'c'
#define K_VOLTAGE               'v'
#define K_ANGLE                 'a'
#define K_APPARENTEPOWER        'A'
#define K_REALPOWER             'r'
#define K_REACTIVEPOWER         'R'
#define K_RAWVOLTAGE            's'

#define VOLT_TO_AMP_FACT        5

#define NRMASK                  0b10000001
#define NR0                     0b10000001
#define NR1                     0b10111101
#define NR2                     0b00010011
#define NR3                     0b00011001
#define NR4                     0b00101101
#define NR5                     0b01001001
#define NR6                     0b01000001
#define NR7                     0b10011101
#define NR8                     0b00000001
#define NR9                     0b00001001

        //shift 1 - 3
#define BIGMASK                 0b110100000000000011100111
#define SMAMASK                 0b111111111111111100011111
#define MASK                    (BIGMASK|SMAMASK)
```

```

#define V                       0b111111111101110101101111
#define WFIRST                  0b111011110101010101111111
#define WSECOND                 0b111111001111111111111111

#define AFIRST                  0b111001110101111001011111
#define ASECOND                 0b111110101111111111111111

#define RSECOND                 0b111111111111111111111111

#define UNIT_NONE               0xffffff
#define UNIT_VA                 (V&ASECOND)
#define UNIT_A                  AFIRST
#define UNIT_W                  WFIRST
#define UNIT_V                  V
#define UNIT_VR                 (V&RSECOND)

#define NNR0                    0b111111101111100011100111
#define NNR1                    0b111111111111111011110111
#define NNR2                    0b111110110111100111100111
#define NNR3                    0b111110110111110011100111
#define NNR4                    0b111110100111111011110111
#define NNR5                    0b111110100111110011101111
#define NNR6                    0b111110100111100011101111
#define NNR7                    0b111111111111111011100111
#define NNR8                    0b111110100111100011100111
#define NNR9                    0b111110100111110011100111

#define WAIT_T0H                NOP();
#define WAIT_T0L                NOP();\
                                NOP();\
                                NOP();
#define WAIT_T1H                NOP();\
                                NOP();\
                                NOP();\
                                NOP();
#define WAIT_T1L                NOP();

#define LED_LOWBIT              {\
                                STATUS_LED=1;\
                                WAIT_T0H\
                                STATUS_LED=0;\
                                WAIT_T0L\
                                }
#define LED_HIGHBIT             {\
                                STATUS_LED=1;\
                                WAIT_T1H\
                                STATUS_LED=0;\
                                WAIT_T1L\
                                }
#define LED_INTENSE              (0xff>>3)

#define DMODE_NONE              0
#define DMODE_CURRENT           1
#define DMODE_VOLTAGE           2
#define DMODE_ANGLE             3
#define DMODE_APPARENT          4
```

```
#define DMODE_REAL              5
#define DMODE_REACTIVE          6
#define DMODE_MAX               7

#define QUARTER_ROTATION           (100)
#define HALF_ROTATION   (QUARTER_ROTATION<<1)
#define FULL_ROTATION   (QUARTER_ROTATION<<2)
#define MIN_SIN_RES     (-100)
#define MAX_SIN_RES     (100)

#define getTime()           TMR1

        /**
         * Prepares the ports of the processor.
         * No device may be turned on. They only "exception" to this rule is the
         * Display, which uses shift registers for storing it's information, it
         * is cleared after all other initialisation steps have been finished.
         * This Method also activates the pull up resistor and sets the operatior
         * frequency to 32MHz.
         */
        void initPins();

        /**
         * Prepare the ADC module for operation. The positive reference is set
         * to Vdd, while the negative one is set to Vss. The conversion clock
         * speed is set to FOSC/64 since the SampleHold - Capacitor would otherwi
         * not be fully charged and unexpected results would be the consequence.
         */
        void initADC();

        /**
         * Prepares the Timer 2 as an refresh-rate generator. This functionality
         * is, as of now, not used and not vital to the operation of the device.
         */
        void initTMR2();

        /**
         * Timer 1 is set up with a resolution of 250ns. It is used to measure th
         * phase delay between Current and Voltage.
         */
        void initTMR1();

        /**
         * initializes both Buffers to 1.024Volts.
         * The first one is needed to measure Vdd with the ADC-Module,
         * the second one is used to provide the comparator with a voltage to
         * compare the Voltage against.
         */
        void initFVR();

        /**
         * Prepares the PWM with Timer 4. This PWM is used to provide the second
         * reference to the second comparator, which is used for the current.
         * Since the voltage, representing the current is small, we need to have
         * a precise reference, this we used a PWM with a low-pass filter of
         * second order to create a direct current.
         * The Output is switched from RB3 to RB0.
```

```
     */
    void initPWMTMR4();

    /**
     * Sets up the Comparator 1 Module for measuring the phase of the Voltage
     * The Interrupt is set to fire on falling edges only.
     */
    void initCOMP1();

    /**
     * Sets up the Comparator 2 Module for measuring the phase of the Current
     * The Interrupt is set to fire on falling edges only.
     */
    void initCOMP2();

    /**
     * Configures the USART Module as asynchronous with an baud rate of 9600
     * and clears all previously received data.
     */
    void initBT();

    /**
     * Performs an AD-Conversion on the specified source. The Sources AN0 to
     * AN11 are proportional to the source specified (setting src to 4 will
     * read from AN4). Also the source for the FVR-Buffer1 can be selected,
     * which is 0x1f;
     *
     * @param src the source from which to convert
     */
    void adc(const int8_t src);

    /**
     * This method is a placeholder method and was written, in case a Voltage
     * measurement was to be implemented. In it's current state it returns th
     * value of 230Volts.
     * @return the line voltage (about 230V in Europe)
     */
    uint8_t readVoltage();

    /**
     * Measures the current which is currently flowing and returns it in mA
     * as Integer to provide an accurate result, without the implications of
     * using floats. The channel from which the measurement is taken is AN7.
     * @return the measured current in mAmps.
     */
    int24_t readCurrent();

    /**
     * Measures Vdd and returns it as an Integer in the range of 0 to 1023.
     * This is useful, because the calculation of the current is a lot easier
     * if first the conversion from all 10 bit values to more reasonable ones
     * is done and only then the currect value computed.
     * @return the supply voltage from 0 to 1023 where 0 = 0V and 1023 = 5V
     */
    uint16_t readVdd();

    /**
```

```
         * Shifts one byte of data into the shift registers with the least
         * significant bit first.
         * @param data the data to write into the shift register
         */
        void so(const uint8_t data);

        /**
         * Clears the display by writing 0xff into every shift register.
         * @param leng the number of registers
         */
        void clearDisplay(int8_t leng);

        /**
         * Sends one byte of colour information to the status led. Since
         * the colour depends on the write order this function does not specify
         * the colour of the LED.
         * @param the intensity of the colour
         */
        void sendColour(uint8_t);

        /**
         * Computes the time difference of the two times. Tm_low is the time, whi
         * came chronologically before tm_high. Since these values are the values
         * of Timer 1 at set time it could be, that tm_low>tm_high, if this is th
         * case the difference will be computed as follos: 0xffff - tm_low + tm_h
         * otherwise the difference is simply tm_high-tm_low.
         * @param tm_low the chronologically first value
         * @param tm_high the chronologically second value
         * @return the time difference in 250nano seconds
         */
        uint16_t deltaT(uint16_t tm_low, uint16_t tm_high);

        /**
         * Reads the sine from the eeprom. It is important to note, that not
         * 360° represent a full rotation, but rather 400°. Since not every
         * value can be stored in the eeprom (it is also not needed) it reads
         * only the value from 0 to 100°.
         * @param z the angle in grad (not deg!)
         * @return the sine multiplied by 100
         */
        int8_t sin_(int8_t z);

        /**
         * Computes values of the sine, which are not covered by sin_().
         * @param z the angle in grad (not deg!)
         * @return the sine multiplied by 100
         */
        int8_t sin(int16_t z);

        /**
         * Behaves in the exactly same way as sin() but returns the cosine.
         * @param z the angle ing grad (not deg!)
         * @return the cosine multiplied by 100
         */
        int8_t cos(int16_t z);

        /**
```

```
         * Evaluates if the LED value can be rewritten (the LED needs a reset
         * time of 50us). If these 50us have passed since the last write to the
         * LED this method returns 1 otherwise 0.
         * @return a flag if the LED can be rewritten
         */
        uint8_t ledReset();

        /**
         * Writes a colour to the LED. The led is programmed with an rgb profile.
         * @param g the green colour intensity
         * @param r the red colour intensity
         * @param b the blue colour intensity
         */
        void setLED(uint8_t g, uint8_t r, uint8_t b);

        /**
         * Writes a specified unit into the shift registers. Note, that all
         * following registers need to be filled, in order for these values to
         * appear in the correct register.
         * @param u the unit to write into the registers
         */
        void setUnit(uint24_t u);

        /**
         * Writes the Integer value into the registers. Typically setUnit() is
         * called prior to this function and only after this function has been
         * called the display will output reasonable values.
         * @param v
         */
        void setVal(int16_t v);

        /**
         * The interrupt service routine.
         * It handles incoming data and the phases of current and voltage.
         */
        void __interrupt ISR();

#ifdef  __cplusplus
}
#endif

#endif
```