```
/*
 * B+O_DAB.c
 *
 * Initial version: 10-07-2023 16:43:32
 * Author : Erik
 * Interface SW for an Attiny861A to read inputs from a Beolit 505
 * and emulate user buttons press on a Karcher 2405 DAB/FM radio.
 * Inputs from the Beolit 505:
 *  AFC/-AFC switch used for selecting DAB/FM on the Karcher 2405
 *  Potentiometer positions for volume level and channel selection.
 * Inputs from the Karcher 2405, during user channel programming:
 *  Button press from: Preset, >, <, Enter
 * Outputs to the Karcher 2405:
 *  Button press emulations for:
 *  Volume-up, volume-down, DAB+/FM, P1..P10
 * 2023.09.03 : Bugfix - internal 1.1V A/D ref. setup corrected
 * 2023.09.03 : Removed internal pull-ups at PORTA5..PORTA2
 * 2023.09.19 : Changing DAB-channel or FM prog. is now momentary
 * 2023.09.23 : Refined volume levels to fit log. volume.potmeter A/D values
 * 2023.10.15 : Using EEprom for saving current DAB/FM state
 *** future option ***
 *   adc [low, high] ranges are set when DAB radio channel programming is
   detected
 *   that is, using the Karchar buttons:  Preset, < , > and Enter buttons
   sequences
 ***
 */
#define F_CPU  119000   // Adjusted 128 KHz nominal CPU clock

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

/*FUSES = {0xE3, 0xDF, 0xFF}; ATtiny861A fuse bytes: Low, High, Extended.
Uses 128 khz internal clock */

#define VolumePin PORTA0          // A/D input pin for volume pot.
#define TunePin PORTA1            // A/D input pin for channel tune pot.
#define PresetPin PORTA2          // Preset button input pin
#define EnterPin PORTA3           // Enter button input pin
#define BackwardPin PORTA4        // Backward button input pin
#define ForwardPin PORTA5         // Forward button input pin
#define DAB_FM_switch_Pin PORTA6  // B&O DAB/FM selector input pin
#define DAB_FM_Pin PORTA7         // Karcher DAB/FM toggle control output pin

#define P1_6 PORTB0       // Karcher P1/P6 select output pin
#define P2_7 PORTB1       // Karcher P2/P7 select output pin
#define P3_8 PORTB2       // Karcher P3/P8 select output pin
#define P4_9 PORTB3       // Karcher P4/P9 select output pin
#define P5_10 PORTB4      // Karcher P5/P10 select output pin
#define Vol_dwPin PORTB5  // Karcher volume down control output pin
#define Vol_upPin PORTB6  // Karcher volume up control output pin
```

```
// PORTB7 is reserved to RESET

// timer0 isr 10 msec. ticks constants for button press active timing
// specifically for the Karcher 2405 DAB+/FM radio button press emulation
#define DAB_FM_ticks 5      // DAB/FM button active ticks
#define Vol_ticks 5         // Volume up/down active ticks
#define P1_P5_ticks 5       // DAB Program 1..5 button active ticks
#define P6_P10_ticks 150    // DAB Program 6..10 button active ticks

#define DAB_ch_select_relax_time 1500 // DAB channel select relax time
#define FM_ch_select_relax_time 1000  // FM channel select relax time
#define EEpromAddr 0x10     // EEprom address for saving current DAB/FM state

void EEpromWrite(unsigned short addr, unsigned char data);
unsigned char EEPROM_read(unsigned char addr);

void DAB_FM_state(void);
void Vol_up(void);
void Vol_dw(void);
void txbyte(uint8_t);
void Set_DABch(uint8_t);
void Set_FMch(uint8_t);

// Channel preset statemachine variables
enum ProgButton { But_None, But_Preset, But_Incr, But_Decr, But_Enter };
enum ProgButton NewButton = But_None;
enum State { Await, Preset, IncrChn, DecrChn, Enter };
enum State ProgState = Await;

static uint8_t Vol_Level = 1, Vol_OldLevel=1, Vol_Adval, Tune_Adval, DAB_ch=11,
  FM_ch=11, Current_DAB_FM_state;
static uint8_t DAB_FM_cnt=0, Volup_ticks=0, Voldw_ticks=0, Tune_ticks=0 ,
  Isr_ch; // future opt. Buttons_old = 0x3C;

// A/D values from tuning pot.meter are in the interval [40..220]
// That is, like using [low, high] as [0,2] will cause change the current DAB -
  channel or FM prog.
// low. and high adc limits for stored DAB channels ( A/D values div. by 2 )
// The values in the DAB_channels array relate to the Beolit 505 Channel scale
  indicators:
//                              5, 10, 15, 20, 25, 30, 35, 40, 45, 50
uint8_t DAB_channels[2][10]={ {103, 95, 87, 78, 71, 63, 55, 47, 40, 32} ,
                              {105, 97, 89, 80, 73, 65, 57, 49, 42, 34} };

// low. and high adc limits for stored FM channels ( A/D MSB div. by 2 )
uint8_t FM_channels[2][10]={ {106, 97, 80, 77, 61, 0, 49, 35, 24, 19} ,
                           {  108, 99, 82, 79, 63, 2, 51, 37, 26, 21} };

int main (void)
{
    DDRB = 0x1F;        // set PORTB7..PORTB5 as inputs, PORTB4..PORTB0 as
      output
    PORTB &= 0xE0;      // set PORTB0..PORTB4 low ( P1/P6 .. P5/P10 )
```

```
    DDRA = 0x80;        // set PORTA7 as output, PORTA6..PORTA0 as inputs
    PORTA &= 0x7F;      // set PORTA7 low ( DAB/FM toggle button )

    /*** future option : bit change isr at the buttons: Preset, < , > and Enter
    GIMSK |= _BV(PCIE1);    //  pin change interrupt at PCINT[7:0]
    PCMSK0 = 0x3C;          // enable pin toggle interrupt at PCINT[2:5]
    */
    // timer0 triggers the 100 Hz timer interrupt routine
    TCCR0A = 0x01;          // enable ctc0
    TCCR0B |= _BV(CS01);    // clk/8
    TIMSK |= _BV(OCIE0A);   // enable output compare a interrupt
    OCR0A = 149;            // 10 msec timer0 interrupt interval

    ADCSRA |= (1<<ADEN); // adc intr enable, adc clock is sysclk prescaled by 2
    ADMUX = 0xA0;   // Intern.ref = 1.1V, 8-bit ADC value, mux ch. = 0
    DIDR0 |= (1<<ADC0D) | (1<<ADC1D) | (1<<AREFD);  // disable digital buffers

    Current_DAB_FM_state = EEPROM_read(EEpromAddr);

    sei();

    _delay_ms(2500);    // Start up delay for the Karcher radio !

    while(1) {
        // determine the new Volume level from the current Adc value
        ADCSRA |= (1<<ADSC);    // restart adc
        while ((ADCSRA & _BV(ADSC)) != 0);  // wait for adc to finish
        if ((ADMUX & 0x01) == 0)            // Ad mux channel = 0 (volume ) ?
        {
            Vol_Adval = ADCH;       // MSB from adc
            if  (Vol_Adval == 0) Vol_Level = 0;
            if ( (Vol_Adval >=1) && (Vol_Adval <= 4) )  Vol_Level = 1;
            if ( (Vol_Adval >=5) && (Vol_Adval <= 9) )  Vol_Level = 2;
            if ( (Vol_Adval >=11) && (Vol_Adval <= 17) )  Vol_Level = 3;
            if ( (Vol_Adval >=19) && (Vol_Adval <= 27) )  Vol_Level = 4;
            if ( (Vol_Adval >=29) && (Vol_Adval <= 45) )  Vol_Level = 5;
            if ( (Vol_Adval >=47) && (Vol_Adval <= 71) )  Vol_Level = 6;
            if ( (Vol_Adval >=73) && (Vol_Adval <= 105) )  Vol_Level = 7;
            if ( (Vol_Adval >=107) && (Vol_Adval <= 143) )  Vol_Level = 8;
            if ( (Vol_Adval >=145) && (Vol_Adval <= 181) )  Vol_Level = 9;
            if ( (Vol_Adval >=183) && (Vol_Adval <= 219) )  Vol_Level = 10;
            if ( (Vol_Adval >=221) && (Vol_Adval <= 255) )  Vol_Level = 11;
            if (Vol_Level > Vol_OldLevel)
            {
                Vol_up();
                _delay_ms(400);
                Vol_OldLevel++;
            }
            if (Vol_OldLevel > Vol_Level)
            {
                Vol_dw();
                _delay_ms(400);
```

```
                Vol_OldLevel--;
            }
            ADMUX |= (1<<MUX0);     // set mux channel to 1 ( tuning ).
        }
        else  // handle adc tuning value
        {
            Tune_Adval = ADCH;
            if  (Current_DAB_FM_state != 0)  // DAB channel in question ?
            {
                Set_DABch(Tune_Adval);
            }
            else
            {
                Set_FMch(Tune_Adval);
            }
            ADMUX &= ~_BV(MUX0);    // set mux channel to  ( volume ).
        }
        DAB_FM_state();
    }
}

void DAB_FM_state(void)
{
    if (Current_DAB_FM_state != 0)
    {
        if ((PINA & _BV(DAB_FM_switch_Pin)) == 0)
        {
            cli();
            DAB_FM_cnt = DAB_FM_ticks;
            PORTA |= _BV(DAB_FM_Pin);   // DAB / FM radio pin high
            Current_DAB_FM_state = 0;   // DAB / FM state = FM
            EEpromWrite(EEpromAddr, 0); // save DAB / FM state
            sei();
            _delay_ms(DAB_ch_select_relax_time);
        }
    }
    else // Current_DAB_FM_state = 0
    {
        if ((PINA & _BV(DAB_FM_switch_Pin)) != 0)
        {
            cli();
            DAB_FM_cnt = DAB_FM_ticks;
            PORTA |= _BV(DAB_FM_Pin);   // DAB / FM radio pin high
            Current_DAB_FM_state = 1;   // DAB / FM state = DAB
            EEpromWrite(EEpromAddr, 1); // save DAB / FM state
            sei();
            _delay_ms(FM_ch_select_relax_time);
        }
    }
}

void Vol_up()
{
```

```
    cli();
    Volup_ticks = Vol_ticks;
    DDRB |= _BV(Vol_upPin);  // Volume up pin as output
    PORTB &= ~_BV(Vol_upPin); // Volume up pin low
    sei();
}

void Vol_dw()
{
    cli();
    Voldw_ticks = Vol_ticks;
    DDRB |= _BV(Vol_dwPin);  // Volume down pin as output
    PORTB &= ~_BV(Vol_dwPin); // Volume down pin low
    sei();
}

void Set_DABch(uint8_t Tune_Advalue)
{
    uint8_t ch_idx=0, new_ch=0;
    // look up channel in DAB adc value range array
    Tune_Advalue = Tune_Advalue/2;  // using half the A/D-value for the DAB
      channel look-up
    while ((ch_idx < 10) && (new_ch == 0))
    {
        if ((Tune_Advalue >= DAB_channels[0][ch_idx] ) && (Tune_Advalue <=
          DAB_channels[1][ch_idx] ))
        {
            if (ch_idx != DAB_ch)
            {
                DAB_ch = ch_idx;
                new_ch = 1;
            }
            else ch_idx = 10;   // stop the while loop, current tuning adc value
               unchanged !
        }
        ch_idx++;       // let the next channel adc range being checked
    }
    if (new_ch != 0)        // new channel to be used
    {
        cli();      // exclude possible timer0 isr variable changes
        PORTB &= 0xE0;      // set PORTB0..PORTB4 low ( P1/P6 .. P5/P10
          deselected )
        _delay_ms(50);      // button deselection relax time
        if (DAB_ch < 5) Tune_ticks = P1_P5_ticks;   // P1..P5 : short button
          active period
        else Tune_ticks = P6_P10_ticks;             // P6..P10 : long button
          active period
        if ((DAB_ch == 0) || (DAB_ch == 5)) PORTB |= _BV(P1_6); // P1 / P6 high
        if ((DAB_ch == 1) || (DAB_ch == 6)) PORTB |= _BV(P2_7); // P2 / P7 high
        if ((DAB_ch == 2) || (DAB_ch == 7)) PORTB |= _BV(P3_8); // P3 / P8 high
        if ((DAB_ch == 3) || (DAB_ch == 8)) PORTB |= _BV(P4_9);  // P4 / P9 high
        if ((DAB_ch == 4) || (DAB_ch == 9)) PORTB |= _BV(P5_10);  // P5 / P10
          high
```

```
        sei();      // allow timer0 isr to start button active timing
    }
}

void Set_FMch(uint8_t Tune_Advalue)
{
    uint8_t ch_idx=0, new_ch=0;
    // look up channel in FM adc value range array
    Tune_Advalue = Tune_Advalue/2;  // using half the A/D-value for the FM
      program look-up
    while ((ch_idx < 10) && (new_ch == 0))
    {
        if ((Tune_Advalue >= FM_channels[0][ch_idx] ) && (Tune_Advalue <=
          FM_channels[1][ch_idx] ))
        {
            if (ch_idx != FM_ch)
            {
                FM_ch = ch_idx;
                new_ch = 1;
            }
            else ch_idx = 10;   // stop the while loop, current tuning adc value
               unchanged !
        }
        ch_idx++;       // let the next channel adc range being checked
    }
    if (new_ch != 0)        // new channel to be used
    {
        cli();      // exclude possible timer0 isr PORTB changes
        PORTB &= 0xE0;      // set PORTB0..PORTB4 low ( P1/P6 .. P5/P10
          deselected )
        _delay_ms(50);      // button deselection relax time
        if (FM_ch < 5) Tune_ticks = P1_P5_ticks;    // P1..P5 : short button
          active period
        else Tune_ticks = P6_P10_ticks;             // P6..P10 : long button
          active period
        if ((FM_ch == 0) || (FM_ch == 5)) PORTB |= _BV(P1_6); // P1 / P6 high
        if ((FM_ch == 1) || (FM_ch == 6)) PORTB |= _BV(P2_7); // P2 / P7 high
        if ((FM_ch == 2) || (FM_ch == 7)) PORTB |= _BV(P3_8); // P3 / P8 high
        if ((FM_ch == 3) || (FM_ch == 8)) PORTB |= _BV(P4_9); // P4 / P9 high
        if ((FM_ch == 4) || (FM_ch == 9)) PORTB |= _BV(P5_10);  // P5 / P10 high
        sei();      // allow timer0 isr to start button active timing
    }
}

ISR(TIMER0_COMPA_vect) {
    if (Volup_ticks != 0)               // Volume up ticks decrement ?
    {
        Volup_ticks--;
        if (Volup_ticks == 0)           // Volume up active timeout ?
        {
            PORTB |= _BV(Vol_upPin);    // Volume up pin high
            DDRB &= ~_BV(Vol_upPin);    // Volume up pin as input
            PORTB &= ~_BV(Vol_upPin);   // Volume up pin pull up
```

```
        }
    }
    if (Voldw_ticks != 0)               // Volume down ticks decrement ?
    {
        Voldw_ticks--;
        if (Voldw_ticks == 0)           // Volume down active timeout ?
        {
            PORTB |= _BV(Vol_dwPin);    // Volume down pin high
            DDRB &= ~_BV(Vol_dwPin);    // Volume down pin as input
            PORTB &= ~_BV(Vol_dwPin);   // Volume down pin pull up
        }
    }
    if (Tune_ticks != 0)                // Tuning ticks decrement ?
    {
        Tune_ticks--;
        if (Tune_ticks == 0)            // Tuning active timeout ?
        {
            if (Current_DAB_FM_state != 0) Isr_ch = DAB_ch;
            else Isr_ch = FM_ch;        // DAB or FM active button time-out ?
            if ((Isr_ch == 0) || (Isr_ch == 5)) PORTB &= ~_BV(P1_6); // P1 / P6 ↵
              down pin low
            if ((Isr_ch == 1) || (Isr_ch == 6)) PORTB &= ~_BV(P2_7); // P2 / P7 ↵
              down pin low
            if ((Isr_ch == 2) || (Isr_ch == 7)) PORTB &= ~_BV(P3_8); // P3 / P8 ↵
              down pin low
            if ((Isr_ch == 3) || (Isr_ch == 8)) PORTB &= ~_BV(P4_9); // P4 / P9 ↵
              down pin low
            if ((Isr_ch == 4) || (Isr_ch == 9)) PORTB &= ~_BV(P5_10);   // P5 / ↵
              P10 down pin low
        }
    }
    if (DAB_FM_cnt != 0)
    {
        DAB_FM_cnt--;
        if (DAB_FM_cnt == 0) PORTA &= ~_BV(DAB_FM_Pin); // DAB / FM radio pin    ↵
          low
    }
}

void EEpromWrite(unsigned short addr, unsigned char data)
{
    while (EECR & (1<<EEPE));       // wait for depending EEPROM write
    EECR = (0<<EEPM1) | (0<<EEPM0); // Enter EEPROM prog. mode
    EEAR = addr;                    // set EEPROM address
    EEDR = data;
    EECR |= (1<<EEMPE);             // enable EEPROM
    EECR |= (1<<EEPE);              // start writing EEPROM
}


unsigned char EEPROM_read(unsigned char addr)
{
    while (EECR & (1<<EEPE));       // wait for depending EEPROM write
```

```
    EEAR = addr;                    // set EEPROM address
    EECR |= (1<<EERE);              //start EEprom read
    return EEDR;                    // Return data
}



/* *** future option
ISR(PCINT_vect)
{
    uint8_t newBitState;
    newBitState = PINA & 0x3C;
    // is it the Increment button being pressed ?
    if ( ((newBitState & _BV(ForwardPin)) == 0) && ((Buttons_old & _BV
      (ForwardPin)) != 0) ) Buttons_old &= ~_BV(ForwardPin);
    else if ( ((newBitState & _BV(ForwardPin)) != 0) && ((Buttons_old & _BV
      (ForwardPin)) == 0) )
    {   // goes here preceding a positive going ForwardPin signal
        Buttons_old |= _BV(ForwardPin);
        NewButton = IncrChn;
    }
    // was it the Decrement button being pressed ?
    if ( ((newBitState & 0x10) == 0) && ((Buttons_old & 0x10) != 0) )
      Buttons_old &= 0xEF;
    else if ( ((newBitState & 0x10) != 0) && ((Buttons_old & 0x10) == 0) )
    {   // goes here preceding a positive going BackwardPin signal
        Buttons_old |= 0x10;
        NewButton = DecrChn;
    }
    // was it the Enter button being pressed ?
    if ( ((newBitState & 0x08) == 0) && ((Buttons_old & 0x08) != 0) )
      Buttons_old &= 0xF7;
    else if ( ((newBitState & 0x08) != 0) && ((Buttons_old & 0x08) == 0) )
    {   // goes here preceding a positive going EnterPin signal
        Buttons_old |= 0x08;
        NewButton = Enter;
    }
    // was it the Preset button being pressed ?
    if ( ((newBitState & 0x04) == 0) && ((Buttons_old & 0x04) != 0) )
      Buttons_old &= 0xFB;
    else if ( ((newBitState & 0x04) != 0) && ((Buttons_old & 0x04) == 0) )
    {   // goes here preceding a positive going PresetPin signal
        Buttons_old |= 0x04;
        NewButton = Preset;
    }
}

//  txbyte(Tune_Advalue);
//  PORTA ^= _BV(TestPin);

void txbyte(uint8_t udbyte)
{
    uint8_t x, bitmask = 0x80;
```

```
    cli();
    PORTA |= _BV(TestPin);
    _delay_us(104);
    for (x=0; x<8; x++)
    {
        if ((udbyte & bitmask) != 0) PORTA |= _BV(TestPin);
        else PORTA &= ~_BV(TestPin);
        bitmask = bitmask >> 1;
        _delay_us(104);
    }
    PORTA &= ~_BV(TestPin);
    _delay_us(104);
    sei();
}
*/
```