This page is part of the mios_c_simulator_-_debugger

ACSim_console.h
ACSim_console.c

ACSim_mios.h
ACSim_mios.c

ACSim_toolbox.h
ACSim_toolbox.c
ACMidiDefines.h

```
/*
 *  ACSim_mios.c
 *  v 0.0.7
 *
 *  2006 April 06 Created by Michael Markert, http://www.audiocommander.de
 *  mios simulator source for MIOS (c) by T. Klose, http://www.midibox.org
 *
 */

/*
 * Released under GNU General Public License
 * http://www.gnu.org/licenses/gpl.html
 *
 * This program is free software; you can redistribute it and/or modify it
under the terms
 * of the GNU General Public License as published by the Free Software
Foundation
 *
 * YOU ARE ALLOWED TO COPY AND CHANGE
 * BUT YOU MUST RELEASE THE SOURCE TOO (UNDER GNU GPL) IF YOU RELEASE YOUR
PRODUCT
 * YOU ARE NOT ALLOWED TO USE IT WITHIN PROPRIETARY CLOSED-SOURCE PROJECTS
 */


#import <Stdio.h>

#import "ACMidiDefines.h"
#import "ACSim_mios.h"
#import "ACSim_toolbox.h"



#pragma mark Globals
// globals
debug_user_timer_t  debug_user_timer;

unsigned char     debug_ain_lastPin;
```

```
unsigned int      debug_ain_value[32];
unsigned char     debug_din_lastPin;
unsigned char     debug_din_value[32];
unsigned int      debug_enc_value[32];

unsigned int      debug_MIDI_byteNum = 0;
unsigned int      debug_IIC_byteNum = 0;

unsigned char     debug_bankstick_ctrl = 0x00;    // 1st Bankstick
preselected
debug_bankstick_t   debug_bankstick[DEBUG_BANKSTICK_NUM];

// "pic18f452.h"
__PORTCbits_t       PORTCbits;
__PORTDbits_t       PORTDbits;
__INTCONbits_t      INTCONbits;
mios_box_stat_t     MIOS_BOX_STAT;
// HLP
unsigned char     MIOS_PARAMETER1;
unsigned char     MIOS_PARAMETER2;
unsigned char     MIOS_PARAMETER3;


#pragma mark MIOS_AIN
// MIOS_AIN
extern void MIOS_AIN_DeadbandSet(unsigned char deadband) { return; }
extern void MIOS_AIN_Muxed(void) { return; }
extern void MIOS_AIN_UnMuxed(void) { return; }
extern void MIOS_AIN_NumberSet(unsigned char pots) { return; }
extern unsigned char MIOS_AIN_Pin7bitGet(unsigned char pin) {
    debug_ain_lastPin = pin;
    // last ainValue is 10bit
    return debug_ain_value[pin] >> 3;
}
extern unsigned int MIOS_AIN_PinGet(unsigned char pin) {
    debug_ain_lastPin = pin;
    return debug_ain_value[pin];
}


#pragma mark MIOS_DIN
// MIOS_DIN
extern unsigned char MIOS_DIN_PinGet(unsigned char pin) {
    debug_din_lastPin = pin;
    if(DEBUG_PEDAL_PRESSED && (pin == DEBUG_BUTTON_AIN_PEDAL)) {
        debug_din_value[pin] = 0;
        return 0; // Button pressed (0V)
    } else {
        return debug_din_value[pin];
    }
}
```

```
#pragma mark MIOS_DOUT
// MIOS_DOUT
extern unsigned char MIOS_SRIO_NumberGet(void) { return 16; }
extern void MIOS_SRIO_NumberSet(unsigned char number_sr) { return; }
extern unsigned char MIOS_SRIO_TS_SensitivityGet(void) { return 0; }
extern void MIOS_SRIO_TS_SensitivitySet(unsigned char update_frq) { return;
}
extern unsigned char MIOS_SRIO_UpdateFrqGet(void) { return 10; }
extern void MIOS_SRIO_UpdateFrqSet(unsigned char update_frq) { return; }
extern unsigned char MIOS_SRIO_DebounceGet(void) { return 32; }
extern void MIOS_SRIO_DebounceSet(unsigned char debounce_value) { return; }


#pragma mark MIOS_ENC
extern void MIOS_ENC_Abs7bitAdd(unsigned char enc, char add) {
    if((debug_enc_value[enc] == 127) && (add > 0)) {
        debug_enc_value[enc] = 127;
    } else if((debug_enc_value[enc] == 0) && (add < 0)) {
        debug_enc_value[enc] = 0;
    } else {
        debug_enc_value[enc] += add;
    }
    return;
}
extern unsigned char MIOS_ENC_Abs7bitGet(unsigned char enc) { return
debug_enc_value[enc]; }
extern void MIOS_ENC_Abs7bitSet(unsigned char enc, unsigned char value) {
debug_enc_value[enc] = value; }
extern unsigned char MIOS_ENC_NumberGet(void) { return 32; }
extern unsigned char MIOS_ENC_SpeedGet(unsigned char enc) { return 0x01; }
extern void MIOS_ENC_SpeedSet(unsigned char enc, unsigned char mode,
unsigned char parameter) { return; }


#pragma mark MIOS_LCD
// MIOS_LCD
extern void MIOS_LCD_Clear(void) { printf("\n"); }
extern void MIOS_LCD_Cmd(unsigned char cmd) { return; }
extern unsigned char MIOS_LCD_CursorGet(void) { return 0; }
extern void MIOS_LCD_CursorSet(unsigned char pos) { printf("\n**MIOS_LCD**
0x%x\t|", pos); }
extern void MIOS_LCD_Data(unsigned char data) { return; }
extern void MIOS_LCD_Init(void) { return; }
extern void MIOS_LCD_PrintBCD1(unsigned char v) { printf("%1.1i",v); }
extern void MIOS_LCD_PrintBCD2(unsigned char v) { printf("%2.2i",v); }
extern void MIOS_LCD_PrintBCD3(unsigned char v) { printf("%3.3i",v); }
extern void MIOS_LCD_PrintBCD4(unsigned int v) { printf("%4.4i",v); }
extern void MIOS_LCD_PrintBCD5(unsigned int v) { printf("%5.5i",v); }
extern void MIOS_LCD_PrintChar(unsigned char c) {
```

```
    switch(c) {
        case 0x00: printf("_(0)"); break;
        case 0x01: printf("_(1)"); break;
        case 0x02: printf("-(2)"); break;
        case 0x03: printf("-(3)"); break;
        case 0x04: printf("-(4)"); break;
        case 0x05: printf("-(5)"); break;
        case 0x06: printf("=(6)"); break;
        case 0x07: printf("=(7)"); break;
        default:
            printf("%c",c);
    }
}
extern void MIOS_LCD_PrintHex1(unsigned char h) { printf("%X",h); }
extern void MIOS_LCD_PrintHex2(unsigned char h) { printf("%X",h); }
extern void MIOS_LCD_PrintPreconfString(unsigned char len, code char
*str[len]) { printf("%s",str); }
extern void MIOS_LCD_PrintString(code char *str) { printf("%s|",str); }
extern void MIOS_LCD_PrintCString(code char *str) { printf("%s|",str); }
extern void MIOS_LCD_PrintMessage(code char *str) { printf("%s|",str); }
extern void MIOS_LCD_MessageStart(unsigned char delay) {
    printf("\n**MIOS_LCD: Output Message delayed for %i ticks", delay);
}
extern void MIOS_LCD_MessageStop(void) { printf("\n**MIOS_LCD: Output
Message stopped"); }
extern void MIOS_CLCD_SpecialCharInit(code char c_table) { return; }
extern void MIOS_CLCD_SpecialCharsInit(code unsigned char *c_table) {
return; }




#pragma mark MIOS_BANKSTICK
// debug bankstick
unsigned char debug_check_bankstick(unsigned int addr) {
    MIOS_BOX_STAT.BS_AVAILABLE = (DEBUG_BANKSTICK_NUM > 0 ? 1 : 0);
    if(! MIOS_BOX_STAT.BS_AVAILABLE) {
        printf("\n**MIOS_BANKSTICK: !!! ERROR !!! : NO BANKSTICK
AVAILABLE!");
        return 1;
    }
    if(addr > DEBUG_BANKSTICK_SIZE) {
        printf("\n**MIOS_BANKSTICK: !!! ERROR !!! : address overflow!");
        //MIOS_BOX_STAT.BS_AVAILABLE = 0;    // this is the default MIOS
behavior on read error
        return 1;
    }
    return 0;    // no error
}

// MIOS_BANKSTICK
```

```
extern unsigned char MIOS_BANKSTICK_CtrlGet(void) { return
debug_bankstick_ctrl; }
extern void MIOS_BANKSTICK_CtrlSet(unsigned char ctrl) {
debug_bankstick_ctrl = ctrl; }

extern unsigned char MIOS_BANKSTICK_Read(unsigned int addr) {
    if(debug_check_bankstick(addr)) { return; }
    unsigned char buf = debug_bankstick[debug_bankstick_ctrl].buffer[addr];
    if(DEBUG_VERBOSE) { printf("\n**MIOS_BANKSTICK_Read(%X): %X", addr,
buf); }
    // clear MIOS_BOX_STAT.BS_AVAILABLE on error
    return buf;
}

extern unsigned char MIOS_BANKSTICK_ReadPage(unsigned int addr, unsigned
char *buffer) {
    if(debug_check_bankstick(addr)) { return; }
    int i;
    for(i=0;i<64;i++) {
        buffer[i] = MIOS_BANKSTICK_Read(addr + i);
    }
    // print buffer
    printf("\n**MIOS_BANKSTICK_ReadPage at 0x%X \n", addr);
    //hexview(buffer, sizeof(buffer));
    hexview(buffer, 64);
    // clear MIOS_BOX_STAT.BS_AVAILABLE on error
    return *buffer;    // return bs_content in readBuffer
}

extern unsigned char MIOS_BANKSTICK_Write(unsigned int addr, unsigned char
value) {
    if(debug_check_bankstick(addr)) { return; }
    debug_bankstick[debug_bankstick_ctrl].buffer[addr] = value;
    if(DEBUG_VERBOSE) { printf("\n**MIOS_BANKSTICK_Write(%X): %X", addr,
value); }
    // errorcodes:    0x00: no error
    //                0x01: byte mismatch (write failed) -- only set if
verify enabled
    //                0x02: BankStick not available
    return 0;    // no error
}

extern unsigned char MIOS_BANKSTICK_WritePage(unsigned int addr, unsigned
char *buffer) {
    if(debug_check_bankstick(addr)) { return; }
    int i;
    for(i=0;i<64;i++) {
        MIOS_BANKSTICK_Write((addr + i), buffer[i]);
    }
    // print buffer
    printf("\n**MIOS_BANKSTICK_WritePage at 0x%X \n", addr);
```

```
    //hexview(buffer, sizeof(buffer));
    hexview(buffer, 64);
    // errorcodes:    0x00: no error
    //                0x01: byte mismatch (write failed) -- only set if
verify enabled
    //                0x02: BankStick not available
    return 0; // no error
}




#pragma mark MIOS_MIDI
// MIOS_MIDI
extern void MIOS_MIDI_BeginStream(void) {
    printf("\n**MIOS_MIDI_BeginStream**");
    debug_MIDI_byteNum = 0;
}
extern void MIOS_MIDI_TxBufferPut(unsigned char b) {
    unsigned char ch;
    printf("\n**MIOS_MIDI_TxBufferPut, %i", b);
    debug_MIDI_byteNum++;
#if DEBUG_MIDI_VERBOSE
    switch(debug_MIDI_byteNum) {
        case 1:
            // Channel Voice
            ch = (b & 0x0f) + 1;
            if( (b>=MIDI_NOTE_OFF) && (b<MIDI_NOTE_ON) )    {
printf("\t\t\tMIDI_NOTE_OFF, CH %i",ch); }
            if( (b>=MIDI_NOTE_ON) && (b<MIDI_POLY_AFTER) )    {
printf("\t\t\tMIDI_NOTE_ON, CH %i",ch); }
            if( (b>=MIDI_POLY_AFTER) && (b<MIDI_CC) )        {
printf("\t\t\tMIDI_POLY_AFTER, CH %i",ch); }
            if( (b>=MIDI_CC) && (b<MIDI_PRG) )               {
printf("\t\t\tMIDI_CC, CH %i",ch); }
            if( (b>=MIDI_PRG) && (b<MIDI_CH_AFTER) )        {
printf("\t\t\tMIDI_PRG, CH %i",ch); }
            if( (b>=MIDI_CH_AFTER) && (b<MIDI_PITCH) )       {
printf("\t\t\tMIDI_CH_AFTER, CH %i",ch); }
            if( (b>=MIDI_PITCH) && (b<MIDI_SYSEX) )          {
printf("\t\t\tMIDI_NOTE_OFF, CH %i",ch); }
            // System Common
            if( (b==MIDI_SYSEX) )                           {
printf("\t\t\tMIDI_SYSEX"); }
            if( (b==MIDI_TIMECODE) )                       {
printf("\t\t\tMIDI_TIMECODE"); }
            if( (b==MIDI_SONG_POSITION_POINTER) )           {
printf("\t\t\tMIDI_SONG_POSITION_POINTER"); }
            if( (b==MIDI_SONG_SELECT) )                     {
printf("\t\t\tMIDI_SONG_SELECT"); }
            if( (b==MIDI_TUNE_REQUEST) )                   {
```

```
printf("\t\t\tMIDI_TUNE_REQUEST"); }
            if( (b==MIDI_EOX) )                                    {
printf("\t\t\tMIDI_END_OF_SYSEX"); }
            // System Realtime
            if( (b==MIDI_CLOCK) )                                  {
printf("\t\t\tMIDI_CLOCK"); }
            if( (b==MIDI_START) )                                  {
printf("\t\t\tMIDI_START"); }
            if( (b==MIDI_CONTINUE) )                               {
printf("\t\t\tMIDI_CONTINUE"); }
            if( (b==MIDI_STOP) )                                   {
printf("\t\t\tMIDI_STOP"); }
            if( (b==MIDI_ACTIVE_SENSING) )                         {
printf("\t\t\tMIDI_ACTIVE_SENSING"); }
            if( (b==MIDI_RESET) )                                  {
printf("\t\t\tMIDI_RESET"); }
            break;
        case 2:
        case 3:
            printf("\t\t\tMIDI_PARAM (BYTE #%i)", b);
            break;
    }
    // reset counter for System Realtime Bytes
    if(b >= MIDI_CLOCK) { debug_MIDI_byteNum = 0; }
#endif
}
extern void MIOS_MIDI_EndStream(void) {
    debug_MIDI_byteNum = 0;
    printf("\n**MIOS_MIDI_EndStream**");
}

extern unsigned char MIOS_MIDI_MergerGet(void) { return 0; }   // not yet
supported!
extern void MIOS_MIDI_MergerSet(unsigned char mode) { return; } // not yet
supported!

/*
 extern void MIOS_MIDI_DeviceIDAutoSet(void) __wparam;
 extern unsigned char MIOS_MIDI_DeviceIDGet(void) __wparam;
 extern void MIOS_MIDI_DeviceIDSet(unsigned char device_id) __wparam;
 extern void MIOS_MIDI_Init(void) __wparam;
 extern unsigned char MIOS_MIDI_InterfaceGet(void) __wparam;
 extern void MIOS_MIDI_InterfaceSet(unsigned char interface) __wparam;
 extern void MIOS_MIDI_InterfaceAutoSet(void) __wparam;
 extern unsigned char MIOS_MIDI_RxBufferFree(void) __wparam;
 extern unsigned char MIOS_MIDI_RxBufferGet(void) __wparam;
 extern void MIOS_MIDI_RxBufferPut(unsigned char b) __wparam;
 extern unsigned char MIOS_MIDI_RxBufferUsed(void) __wparam;
 extern void MIOS_MIDI_TxBufferFlush(void) __wparam;
 extern unsigned char MIOS_MIDI_TxBufferFree(void) __wparam;
 extern unsigned char MIOS_MIDI_TxBufferGet(void) __wparam;
```

```
 extern void MIOS_MIDI_TxBufferPut(unsigned char b) __wparam;
 extern unsigned char MIOS_MIDI_TxBufferUsed(void) __wparam;
 */

#pragma mark MIOS_TIMER
// MIOS_TIMER
extern void MIOS_TIMER_Init(unsigned char mode, unsigned int period) {
    debug_user_timer.TIMER_ENABLED = TRUE;
    return;
}
extern void MIOS_TIMER_Start(void) { return; }
extern void MIOS_TIMER_Stop(void) {
    debug_user_timer.TIMER_ENABLED = FALSE;
    return;
}
extern void MIOS_TIMER_ReInit(unsigned char mode, unsigned int period) {
    MIOS_TIMER_Init(mode, period);
    return;
}


#pragma mark MIOS_IIC
// MIOS_IIC
extern void MIOS_IIC_Start(void) {
    debug_IIC_byteNum = 0;
    printf("\n**MIOS_IIC_Start**");
}
extern void MIOS_IIC_Stop(void) {
    printf("\n**MIOS_IIC_Stop**");
}
extern unsigned char MIOS_IIC_ByteSend(unsigned char b)  {
    printf("\n**MIOS_IIC_ByteSend: 0x%x  \t %i  \t %c", b, b, b);
#if DEBUG_SPEAKJET_VERBOSE
    if(debug_IIC_byteNum == 1) {
        // MSA controls
        switch(b) {
            case 0: case 1: case 2: case 3: case 4: case 5: case 6: printf("
\t MSA_PAUSE");    break;
            case 7: printf(" \t MSA_NEXTFAST");
break;
            case 8: printf(" \t MSA_NEXTSLOW");
break;
            case 14: printf(" \t MSA_NEXTHIGH");
break;
            case 15: printf(" \t MSA_NEXTLOW");
break;
            case 16: printf(" \t MSA_WAIT");
break;
            case 20: printf(" \t MSA_VOLUME");
break;
```

```
            case 21: printf(" \t MSA_SPEED");
break;
            case 22: printf(" \t MSA_PITCH");
break;
            case 23: printf(" \t MSA_BEND");
break;
            case 24: printf(" \t MSA_PORTCTR");
break;
            case 25: printf(" \t MSA_PORT");
break;
            case 26: printf(" \t MSA_REPEAT");
break;

            case 28: printf(" \t MSA_CALLPHRASE");
break;
            case 29: printf(" \t MSA_GOTOPHRASE");
break;
            case 30: printf(" \t MSA_DELAY");
break;
            case 31: printf(" \t MSA_RESET");
break;

            default:
                if(b > 127) {
                    if(b < 200) {
                        printf(" \t MSA_SOUNDCODE_PHONEME");
                    } else if(b < 255) {
                        printf(" \t MSA_SOUNDCODE_FX");
                    } else {
                        printf(" \t MSA unknown: %i", b);
                    }
                }
                break;
        }
    }
    // SCP controls
    switch(b) {
        case SCP_ESCAPE:        printf(" \t --- ENTER SCP MODE ---");
break;
        case SCP_EXIT:            printf(" \t --- LEAVE SCP MODE ___");
break;
        case SCP_READY:            printf(" \t READY");
break;
        case SCP_CLEAR_BUFFER:    printf(" \t CLEAR BUFFER");
break;
        case SCP_START:            printf(" \t START");
break;
        case SCP_STOP:            printf(" \t STOP");
break;
        case SCP_MEMTYPE:        printf(" \t  ^  MEMTYPE (REG:0 EEPROM_L:2
EEPROM_H:3)");        break;
```

```
        case SCP_MEMADDR:         printf(" \t  ^  MEMADDR (freq:1-5 lvl:11-15
dist:6, vol:7)");    break;
        case SCP_MEMWRT:        printf(" \t  ^  STORED");
break;
        case SCP_RESET:           printf(" \t RESET");
break;
    }
#endif
    debug_IIC_byteNum++;
    return 1;
}
extern unsigned char MIOS_IIC_ByteReceive(void) {
    printf("\n**MIOS_IIC_ByteReceived**");
    return 1;
}
extern void MIOS_IIC_AckSend(void)             { printf("\n**MIOS_IIC_ACK
sent**"); }
extern void MIOS_IIC_NakSend(void)             { printf("\n**MIOS_IIC_NAK
sent**"); }
extern void MIOS_IIC_CtrlSet(unsigned char ctrl){ return; }
extern unsigned char MIOS_IIC_CtrlGet(void)       {
    return 0;    // not sure about this return value!
}


#pragma mark MIOS_HELP
extern unsigned char MIOS_HLP_GetBitANDMask(unsigned char value) {
    switch(value) {
        case 0:    return 0xFE;    break;
        case 1: return 0xFD;    break;
        case 2: return 0xFB;    break;
        case 3: return 0xF7;    break;
        case 4: return 0xEF;    break;
        case 5: return 0xDF;    break;
        case 6: return 0xBF;    break;
        case 7: return 0x7F;    break;
        default:
            return 0;
            break;
    }
}
extern unsigned char MIOS_HLP_GetBitORMask(unsigned char value) {
    switch(value) {
        case 0:    return 0x1;        break;
        case 1: return 0x2;        break;
        case 2: return 0x4;        break;
        case 3: return 0x8;        break;
        case 4: return 0x10;    break;
        case 5: return 0x20;    break;
        case 6: return 0x40;    break;
        case 7: return 0x80;    break;
```

```
        default:
            return 0;
            break;
    }
}
extern unsigned char MIOS_HLP_16bitAddSaturate(char add_value, unsigned int
*ptr, unsigned int max_value) {
    // saturate to 0 or MAX
    // returns 0 if saturated, 1 if in-/decreased
    int i = (int)(*ptr + add_value);
    if(i < 0) {
        *ptr = 0;
        return 0;
    } else if(i > (signed)max_value) {
        *ptr = (unsigned char)max_value;
        return 0;
    } else {
        *ptr = (unsigned char)i;
        return 1;
    }
}
extern void MIOS_HLP_Dec2BCD(unsigned int value) {
    // decimals to hex (eg 123456 to 0x12, 0x34, 0x56)
    MIOS_PARAMETER1 = 0x0;
    MIOS_PARAMETER2 = 0x0;
    MIOS_PARAMETER3 = 0x0;
    if(value > 999999) {
        printf("\nMIOS_HLP_Dec2BCD_ERROR: value exceeds 6 digits!
ABORTING**");
        return;
    }
    unsigned char a;
    unsigned char b;
    // get 100.000
    a = (unsigned char)(value / 100000);
    b = (unsigned char)((value / 10000) - (a * 10));
    MIOS_PARAMETER3 = (a << 4) + b;
    // get 1.000
    value = value - (a * 100000 + b * 10000);
    a = (unsigned char)(value / 1000);
    b = (unsigned char)((value / 100) - (a * 10));
    MIOS_PARAMETER2 = (a << 4) + b;
    // get 10
    value = value - (a * 1000 + b * 100);
    a = (unsigned char)(value / 10);
    b = (unsigned char)(value - (a * 10));
    MIOS_PARAMETER1 = (a << 4) + b;
}
```