```
#include <xs1.h>
#include <platform.h>
#include <stdio.h>
#include <syscall.h>
#include <xclib.h>
#include <debug_print.h>

#include "devicedefines.h"
#include "xc_ptr.h"
#include "audio.h"


/* audio I/O */
extern          in port     p_bitclk;           // MCLK = bit clock
extern buffered in port:32  p_frameclk;         //
extern buffered in port:32  p_tdm_in[3];        // 3 * I16S
extern buffered out port:32 p_tdm_out[3];       // 3 * I16S

/* audio clock */
extern clock                clk_audio_bclk;     // driven by
p_bitclk

//
int32_t audio_buf_0[NUM_USB_CHAN_OUT + NUM_USB_CHAN_IN];   //
outputs + inputs
int32_t audio_buf_1[NUM_USB_CHAN_OUT + NUM_USB_CHAN_IN];   //


/* 2nd decouple stage between usb core decouple and audio port
driver */
// void audio_decouple(chanend c_out, server interface if_dbl_buf
if_audio, unsigned device_config)
void audio_decouple(chanend c_out, chanend c_cfg, server interface
if_dbl_buf if_audio)
{
    unsigned sync_flag;
    unsigned sample;
    unsigned device_config;
    unsigned chan_count_out;
    unsigned chan_count_in;

    int32_t * movable mp_buffer = &audio_buf_0[0];

    // get config from Endpoint0()
    device_config = inuint(c_cfg);

    // debug_printf("device_config = %x\n", device_config);
    chan_count_out = (device_config >> 16) & 0xFF;  //
NUM_USB_CHAN_OUT;
    chan_count_in = (device_config >> 24) & 0xFF;   //
NUM_USB_CHAN_IN;

    //
    while(1)
```

```
    {
        select
        {
            case if_audio.swap(int32_t * movable &mp_new):

                    /* swap movable pointers */
                    int32_t * movable mp_tmp = move(mp_new);
                    mp_new = move(mp_buffer);
                    mp_buffer = move(mp_tmp);
                    /* set sync flag to trigger signal processing
*/
                    sync_flag = 1;

                    break;

             default:
                    if(sync_flag)
                    {
                        /* sync with decouple() - command =
deliver(c_mix_out); */
                        outuint(c_out, 0);

#pragma loop unroll
                        for(unsigned i = 0; i < NUM_USB_CHAN_OUT;
i++)
                        {
                            // mp_buffer[i] =
bitrev(inuint(c_out));
                            sample = bitrev(inuint(c_out));
                            if(i < chan_count_out)
                                mp_buffer[i] = sample;
                            else
                                mp_buffer[i] = 0;
                        }

#pragma loop unroll
                        for(unsigned i = 0; i < NUM_USB_CHAN_IN; i+
+)
                        {
                            // outuint(c_out,
bitrev(mp_buffer[NUM_USB_CHAN_OUT + i]));
#if USB_LOOPBACK
                            sample = bitrev(mp_buffer[i]);
#else
                            sample =
bitrev(mp_buffer[NUM_USB_CHAN_OUT + i]);
#endif
                            if(i < chan_count_in)
                                outuint(c_out, sample);
                            else
                                outuint(c_out, 0);
                        }

                        /* reset sync flag */
```

```
                        sync_flag = 0;

                    }
                    break;
        }
    }
}


//
#define SYNC_COMPARE   0x7FFFFFFF

//
unsigned i16s_in_buf [48 << 1];     // double buffer
unsigned i16s_out_buf[48 << 1];     //
unsigned resync_cnt = 0;

//
void audio_port_driver(client interface if_dbl_buf if_audio)
{
    unsigned sync_errors = 0;
    unsigned resync_flag;
    unsigned fsync_data;
    unsigned fsync_buf[16];
    unsigned timeslot;
    unsigned portclk;
    unsigned tmp;

    unsigned adc_tmp[48];

    // initialize movable pointer (for double buffering)
    int32_t * movable mp_audio = &audio_buf_1[0];

    //
    // debug_printf("start I16S transceiver ...\n");

    //
    set_thread_fast_mode_on();

    // configure_audio_ports_for_streaming();
    // wait until FSYNC is low
     tmp = 1;
     while (tmp)
        p_frameclk :> tmp;

    stop_clock(clk_audio_bclk);

    clearbuf(p_tdm_out[0]);
    clearbuf(p_tdm_out[1]);
    clearbuf(p_tdm_out[2]);
    clearbuf(p_tdm_in[0]);
    clearbuf(p_tdm_in[1]);
    clearbuf(p_tdm_in[2]);
    clearbuf(p_frameclk);
```

```
    set_port_inv(p_bitclk);
    set_pad_delay(p_bitclk, 5);     // 10 ns
    set_port_no_inv(p_frameclk);    //

    // generate bit clock block from pin - TDM: bit clock = master
clock
    configure_clock_src(clk_audio_bclk, p_bitclk);
    // clock output data ports from clock block
    // output ports are shifted at falling (internal) clock edges
    configure_out_port_no_ready(p_tdm_out[0], clk_audio_bclk, 0);
    configure_out_port_no_ready(p_tdm_out[1], clk_audio_bclk, 0);
    configure_out_port_no_ready(p_tdm_out[2], clk_audio_bclk, 0);
    // clock input data ports from clock block
    // inputs are sampled at rising (internal) clock edges
    configure_in_port_no_ready(p_tdm_in[0], clk_audio_bclk);
    configure_in_port_no_ready(p_tdm_in[1], clk_audio_bclk);
    configure_in_port_no_ready(p_tdm_in[2], clk_audio_bclk);
    configure_in_port_no_ready(p_frameclk,  clk_audio_bclk);
    // shift sampling of input ports to falling (internal) clock
edges
    set_port_sample_delay(p_tdm_in[0]);
    set_port_sample_delay(p_tdm_in[1]);
    set_port_sample_delay(p_tdm_in[2]);
    set_port_sample_delay(p_frameclk);

    start_clock(clk_audio_bclk);

    //
    resync_flag = 1;

    // wait for audio clocks stable
    fsync_data = 0;
    for (int i = 0; i < 50; i++)
    {
        while (!fsync_data)         // wait for rising FSYNC edge
            p_frameclk :> fsync_data;
        while (fsync_data)          // wait for falling FSYNC edge
            p_frameclk :> fsync_data;
    }

    // audio driver loop ( init/reinit - loop)
    while (1)
    {
        // init
        if (resync_flag)
        {
            resync_flag = 0;
            sync_errors = 0;
            resync_cnt ++;

#if 0
            // wait until MCLK is low
            tmp = 1;
```

```
            while (tmp)
                p_mclk :> tmp;
#endif

            stop_clock(clk_audio_bclk);

            clearbuf(p_tdm_out[0]);
            clearbuf(p_tdm_out[1]);
            clearbuf(p_tdm_out[2]);
            clearbuf(p_tdm_in[0]);
            clearbuf(p_tdm_in[1]);
            clearbuf(p_tdm_in[2]);
            clearbuf(p_frameclk);

            start_clock(clk_audio_bclk);

/*
            // wait for fsync = HIGH
            fsync_data = 0;
            while (fsync_data != 0xffffffff)
                p_frameclk :> fsync_data;

            // catch falling FSYNC edge
            while (fsync_data == 0xffffffff)
                p_frameclk :> fsync_data @ portclk;

           // calculate delay to start data streaming in sync to
frame start (falling edge of FSYNC)
            delay_in = portclk + 4096;          // has to be a
multiple of 512 to match both formats (I8S / I16S)

            // 48 KHz
            delay_in += 448 - 1;            // align to 2 samples
before frame sync

            while (fsync_data != 0x7FFFFFFF)
            {
                delay_in--;
                fsync_data <<= 1;               // fsync_data is
bit-reversed...
                fsync_data |= 1;
            }

            delay_out = delay_in + 32 + 1;

            // timed in sample 0 -2
            p_tdm_in[0] @ (delay_in) :> void;   // synced dummy
inputs
            p_tdm_in[1] @ (delay_in) :> void;
            p_tdm_in[2] @ (delay_in) :> void;
            p_frameclk  @ (delay_in) :> fsync_data;

            // timed out
            p_tdm_out[0] @ (delay_out) <: 0xffffffff;
```

```
            p_tdm_out[1] @ (delay_out) <: 0xffffffff;
            p_tdm_out[2] @ (delay_out) <: 0xffffffff;

            p_tdm_in[0] :> void;                // synced dummy
inputs
            p_tdm_in[1] :> void;
            p_tdm_in[2] :> void;
            p_frameclk  :> void;
*/

            p_frameclk when pinseq(1) :> void;
            p_frameclk when pinseq(0) :> void @ portclk;

            unsigned initial_out_port_time = portclk - 32 + (16*32);
            unsigned initial_in_port_time  = portclk - 32 +
(16*32)+32 - 1;

            p_tdm_out[0] @ initial_out_port_time <: mp_audio[15];
            p_tdm_out[1] @ initial_out_port_time <: mp_audio[31];
            p_tdm_out[2] @ initial_out_port_time <: mp_audio[47];

            // XC doesn't have syntax for setting a timed input
without waiting for the input
            asm("setpt res[%0],
%1"::"r"(p_frameclk),"r"(initial_in_port_time));

            asm("setpt res[%0],
%1"::"r"(p_tdm_in[0]),"r"(initial_in_port_time));
            asm("setpt res[%0],
%1"::"r"(p_tdm_in[1]),"r"(initial_in_port_time));
            asm("setpt res[%0],
%1"::"r"(p_tdm_in[2]),"r"(initial_in_port_time));
        }

        /* 48 KHz / 48 ch - I16S */
        for (timeslot = 0; timeslot < 16; timeslot++)
        {

#if AUDIO_LOOPBACK
            p_tdm_out[0] <: mp_audio[NUM_USB_CHAN_OUT + timeslot +
0];
            p_tdm_out[1] <: mp_audio[NUM_USB_CHAN_OUT + timeslot +
16];
            p_tdm_out[2] <: mp_audio[NUM_USB_CHAN_OUT + timeslot +
32];
#else
            p_tdm_out[0] <: mp_audio[timeslot +  0];
            p_tdm_out[1] <: mp_audio[timeslot + 16];
            p_tdm_out[2] <: mp_audio[timeslot + 32];
#endif

            p_frameclk  :> fsync_buf[timeslot]; // fsync_data;

            if (timeslot == 0)
```

```
            {
                p_tdm_in[0] :> mp_audio[NUM_USB_CHAN_OUT + timeslot
+ 15];
                p_tdm_in[1] :> mp_audio[NUM_USB_CHAN_OUT + timeslot
+ 31];
                p_tdm_in[2] :> mp_audio[NUM_USB_CHAN_OUT + timeslot
+ 47];
            }
            else
            {
                p_tdm_in[0] :> adc_tmp[timeslot -  1];
                p_tdm_in[1] :> adc_tmp[timeslot + 15];
                p_tdm_in[2] :> adc_tmp[timeslot + 31];
            }

            //
            if (timeslot < 15) {
                /* fill audio double buffer with buffered adc values
*/
                mp_audio[NUM_USB_CHAN_OUT + timeslot +  0] =
adc_tmp[timeslot +  0];
                mp_audio[NUM_USB_CHAN_OUT + timeslot + 16] =
adc_tmp[timeslot + 16];
                mp_audio[NUM_USB_CHAN_OUT + timeslot + 32] =
adc_tmp[timeslot + 32];
            }
        }

        // make sure serial ports are still in sync with FSYNC
        if (fsync_buf[0] != 0x7FFFFFFF)
        {
            // wrong alignment to FSYNC
            sync_errors++;
            if (sync_errors > 5)
            {
                resync_flag = 1;
                debug_printf("resync = %d\n", resync_cnt);
            }
        }

        /* swap audio buffer pointers (requires about 300 ns) */
        if_audio.swap(mp_audio);

    }   // while(1)

}
```