codec.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity clkgen is
    generic
    (
        CHANNEL_DURATION: positive := 128    -- must be 128
    );
    port
    (
        -- interface I/O signals
        clk: in std_logic;       -- clock input
        sreset: in std_logic;    -- synchronous active-high reset
        areset: in std_logic;    -- asynchronous active-high reset
        -- codec chip clock signals
        mclk: out std_logic;     -- master clock output to codec
        sclk: out std_logic;     -- serial data clock to codec
        lrck: out std_logic;     -- left/right codec channel select
        bit_cntr: out std_logic_vector(5 downto 0);
        subcycle_cntr: out std_logic_vector(1 downto 0)
    );
end clkgen;

architecture clkgen_arch of clkgen is
signal lrck_int: std_logic;
signal seq: std_logic_vector(7 downto 0);
begin
    gen_clock:
    process(clk,areset,seq,lrck_int)
    begin
        if(areset='0') then -- synchronous reset
            seq <= (others=>'0');
            lrck_int <= LEFT;    -- start with left channel of codec
        elsif (clk'event and clk='1') then
            if(sreset='0') then -- synchronous reset
                seq <= (others=>'0');
                lrck_int <= LEFT;    -- start with left channel of codec
            elsif(seq=CHANNEL_DURATION-1) then
                seq <= (others=>'0');    -- reset sequencer after every channel period
                lrck_int <= not(lrck_int);   -- toggle channel selector every period
            else
                seq <= seq+1;
                lrck_int <= lrck_int;
            end if;
        end if;
    end process;
    lrck <= lrck_int;    -- output the channel selector to the codec
    mclk <= clk;         -- codec master clock equals input clock
    sclk <= seq(1);      -- the serial data shift clock is 1/4 of the master clock
    bit_cntr <= seq(7 downto 2);
```

```
    subcycle_cntr <= seq(1 downto 0);
end clkgen_arch;




library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity channel is
    generic
    (
        XSTEND_V1_2  : boolean  := false;   -- XStend V1.3 uses AKM codec; V1.2 uses Crystal codec
        DAC_WIDTH: positive := 20;
        ADC_WIDTH: positive := 20
    );
    port
    (
        -- interface I/O signals
        clk: in std_logic;       -- clock input
        sreset: in std_logic;    -- synchronous active-high reset
        areset: in std_logic;    -- asynchronous active-high reset
        chan_on: in std_logic;   -- one if channel is communicating with codec
        bit_cntr: in std_logic_vector(5 downto 0);
        subcycle_cntr: in std_logic_vector(1 downto 0);
        chan_sel: in std_logic; -- select the left or right codec channel for reading or writing
        rd: in std_logic;            -- read from the codec ADC
        wr: in std_logic;            -- write to the codec DAC
        adc_out: out std_logic_vector(ADC_WIDTH-1 downto 0);      -- output from codec ADC
        dac_in: in std_logic_vector(DAC_WIDTH-1 downto 0);        -- input to codec DAC
        adc_out_rdy: out std_logic;      -- ADC output is ready to be read
        adc_overrun: out std_logic;      -- output from ADC channel overwritten before being read
        dac_in_rdy: out std_logic;       -- DAC input is ready to be written
        dac_underrun: out std_logic;     -- input to DAC did not arrive in time
        -- codec chip I/O signals
        sdin: out std_logic;     -- serial output to codec DAC
        sdout: in std_logic      -- serial input from codec ADC
    );
end channel;

architecture channel_arch of channel is
signal dac_shfreg: std_logic_vector(DAC_WIDTH-1 downto 0);   -- DAC shift register
signal dac_empty: std_logic;     -- DAC shift register is empty
signal dac_wr: std_logic;    -- the DAC channel has been written
signal dac_wr_nxt: std_logic;    -- the DAC channel has been written
signal dac_in_rdy_int: std_logic;    -- internal version of DAC is ready for input signal
signal adc_shfreg: std_logic_vector(ADC_WIDTH-1 downto 0);   -- ADC shift register
signal adc_full: std_logic; -- ADC shift register is full
signal adc_rd: std_logic;    -- the ADC channel has been read
signal adc_rd_nxt: std_logic;    -- the ADC channel has been read
```

```
signal adc_out_rdy_int: std_logic;  -- internal version of ADC output is ready signal
signal start_bit: integer;
signal end_bit: positive;
begin
    -- setup the bit slot bounds for the data to/from the codec
    start_bit <= 0           when XSTEND_V1_2=false else 1;
    end_bit   <= ADC_WIDTH-1 when XSTEND_V1_2=false else ADC_WIDTH;

    -- receives data from codec ADC
    rcv_adc:
    process(clk,areset,chan_on,subcycle_cntr,bit_cntr,adc_shfreg,sdout)
    begin
        if(areset='0') then
            adc_shfreg <= (others=>'0');
            adc_full <= NO;
        elsif(clk'event and (clk=YES)) then
            if(sreset='0') then
                adc_shfreg <= (others=>'0');
                adc_full <= NO;
            elsif((chan_on=YES) and (subcycle_cntr=2)) then
                if((bit_cntr>=start_bit) and (bit_cntr<end_bit)) then
                    adc_full <= NO;
                    adc_shfreg <= adc_shfreg(ADC_WIDTH-2 downto 0) & sdout;
                elsif(bit_cntr=end_bit) then
                    adc_full <= YES;
                    adc_shfreg <= adc_shfreg(ADC_WIDTH-2 downto 0) & sdout;
                end if;
            end if;
        end if;
    end process;
    adc_out <= adc_shfreg;

    -- handle reading of ADC data from codec interface
    adc_rd_nxt <= YES when (adc_full=YES and chan_sel=YES and rd=YES) or
                                          (adc_full=YES and adc_rd=YES)
                      else NO;
    read_adc:
    process(clk,areset,adc_rd_nxt)
    begin
        if(areset='0') then
            adc_rd <= NO;
        elsif(clk'event and clk='1') then
            if(sreset='0') then
                adc_rd <= NO;
            else
                adc_rd <= adc_rd_nxt;
            end if;
        end if;
    end process;
    -- ADC data is ready for reading if register is full and hasn't been read yet
    adc_out_rdy_int <= YES when adc_full=YES and adc_rd=NO else NO;
    adc_out_rdy <= adc_out_rdy_int;
```

```
    -- detect and signal overwriting of data from the codec ADC channels
    detect_adc_overrun:
    process(clk,areset,bit_cntr,chan_on,adc_out_rdy_int)
    begin
        if(areset='0') then
            adc_overrun <= NO;
        elsif(clk'event and clk='1') then
            if(sreset='0') then
                adc_overrun <= NO;
            elsif(bit_cntr=1 and chan_on=YES and adc_out_rdy_int=YES) then
                adc_overrun <= YES;
            end if;
        end if;
    end process;

    -- transmits data to codec DAC
    tx_dac:
    process(clk,areset,chan_on,subcycle_cntr,bit_cntr,dac_shfreg)
    begin
        if(areset='0') then
            dac_shfreg <= (others=>'0');
            dac_empty <= YES;
        elsif(clk'event and clk='1') then
            if(areset='0') then
                dac_shfreg <= (others=>'0');
                dac_empty <= YES;
            elsif(chan_sel=YES and wr=YES) then
                dac_shfreg <= dac_in;
            elsif(chan_on=YES and subcycle_cntr=2) then
                if((bit_cntr>=start_bit) and (bit_cntr<end_bit)) then
                    dac_empty <= NO;
                    dac_shfreg <= dac_shfreg(DAC_WIDTH-2 downto 0) & '0';
                elsif(bit_cntr=end_bit) then
                    dac_empty <= YES;
                    dac_shfreg <= dac_shfreg(DAC_WIDTH-2 downto 0) & '0';
                end if;
            end if;
        end if;
    end process;

    -- output the serial data to the SDIN pin of the codec DAC
    sdin <= dac_shfreg(DAC_WIDTH-1) when chan_on=YES else '0';

    -- handle writing of DAC data from codec interface
    dac_wr_nxt <= YES when (dac_empty=YES and chan_sel=YES and wr=YES) or
                                                    (dac_empty=YES and dac_wr=YES)
                        else NO;
    write_dac:
    process(clk,areset,dac_wr_nxt)
    begin
        if(areset='0') then
```

```
            dac_wr <= NO;
        elsif(clk'event and clk='1') then
            if(sreset='0') then
                dac_wr <= NO;
            else
                dac_wr <= dac_wr_nxt;
            end if;
        end if;
    end process;
    -- DAC is ready for writing if register is empty and hasn't been written yet
    dac_in_rdy_int <= YES when dac_empty=YES and dac_wr=NO else NO;
    dac_in_rdy <= dac_in_rdy_int;

    -- detect and signal underflow of data to the codec DAC channels
    detect_dac_underrun:
    process(clk,areset,bit_cntr,chan_on,dac_in_rdy_int)
    begin
        if(areset='0') then
            dac_underrun <= NO;
        elsif(clk'event and clk='1') then
            if(sreset='0') then
                dac_underrun <= NO;
            elsif(bit_cntr=1 and chan_on=YES and dac_in_rdy_int=YES) then
                dac_underrun <= YES;
            end if;
        end if;
    end process;
end channel_arch;




library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.codec.all;

entity codec_intfc is
    generic
    (
        XSTEND_V1_2  : boolean  := false;  -- XStend V1.3 uses AKM codec; V1.2 uses Crystal codec
        DAC_WIDTH: positive := 20;
        ADC_WIDTH: positive := 20;
        CHANNEL_DURATION: positive := 128    -- must be 128
    );
    port
    (
        -- interface I/O signals
        clk: in std_logic;       -- clock input
        sreset: in std_logic;    -- synchronous active-high reset
        areset: in std_logic;    -- asynchronous active-high reset
```

```
        lrsel: in std_logic;     -- select the left or right codec channel for reading or writing
        rd: in std_logic;            -- read from the codec ADC
        wr: in std_logic;            -- write to the codec DAC
        ladc_out: out std_logic_vector(ADC_WIDTH-1 downto 0);    -- output from codec left-channel ADC
        radc_out: out std_logic_vector(ADC_WIDTH-1 downto 0);    -- output from codec right-channel ADC
        ldac_in: in std_logic_vector(DAC_WIDTH-1 downto 0);      -- input to codec left-channel DAC
        rdac_in: in std_logic_vector(DAC_WIDTH-1 downto 0);      -- input to codec right-channel DAC
        ladc_out_rdy: out std_logic;     -- left-channel ADC output is ready to be read
        radc_out_rdy: out std_logic;     -- right-channel ADC output is ready to be read
        adc_overrun: out std_logic;      -- output from ADC channel overwritten before being read
        ldac_in_rdy: out std_logic;      -- left channel DAC input is ready to be written
        rdac_in_rdy: out std_logic;      -- right-channel DAC input is ready to be written
        dac_underrun: out std_logic;     -- DAC did not receive input data in time
        -- codec chip I/O signals
        mclk: out std_logic;     -- master clock output to codec
        sclk: out std_logic;     -- serial data clock to codec
        lrck: out std_logic;     -- left/right codec channel select
        sdin: out std_logic;     -- serial output to codec DAC
        sdout: in std_logic      -- serial input from codec ADC
    );
end codec_intfc;

architecture codec_intfc_arch of codec_intfc is
signal mclk_int: std_logic; -- internal codec master clock
signal lrck_int: std_logic; -- internal left/right codec channel select
signal sclk_int: std_logic; -- internal codec data shift clock
signal sdin_int: std_logic; -- internal data stream to the codec DAC
signal bit_cntr: std_logic_vector(5 downto 0);
signal subcycle_cntr: std_logic_vector(1 downto 0);
signal lsdin: std_logic;
signal rsdin: std_logic;
signal ladc_overrun: std_logic;
signal radc_overrun: std_logic;
signal ldac_underrun: std_logic;
signal rdac_underrun: std_logic;
signal lchan_sel: std_logic;
signal rchan_sel: std_logic;
signal lchan_on: std_logic;
signal rchan_on: std_logic;
begin

    u0: clkgen
            generic map
            (
                CHANNEL_DURATION=>CHANNEL_DURATION
            )
            port map
            (
                clk=>clk,
                sreset=>sreset,
                areset=>areset,
                mclk=>mclk_int,
```

```
            sclk=>sclk_int,
            lrck=>lrck_int,
            bit_cntr=>bit_cntr,
            subcycle_cntr=>subcycle_cntr
        );

    -- invert signals on XStend V1.3, but not on XStend V1.2
    lrck <= not(lrck_int) when XSTEND_V1_2=false else lrck_int;
    mclk <= not(mclk_int) when XSTEND_V1_2=false else mclk_int;
    sclk <= not(sclk_int) when XSTEND_V1_2=false else sclk_int;

    lchan_sel <= YES when lrsel=LEFT else NO;
    lchan_on <= YES when lrck_int=LEFT else NO;
    u_left: channel
        generic map
        (
            XSTEND_V1_2=>XSTEND_V1_2,
            DAC_WIDTH=>DAC_WIDTH,
            ADC_WIDTH=>ADC_WIDTH
        )
        port map
        (
            clk=>clk,
            sreset=>sreset,
            areset=>areset,
            chan_on=>lchan_on,
            bit_cntr=>bit_cntr,
            subcycle_cntr=>subcycle_cntr,
            chan_sel=>lchan_sel,
            rd=>rd,
            wr=>wr,
            adc_out=>ladc_out,
            dac_in=>ldac_in,
            adc_out_rdy=>ladc_out_rdy,
            adc_overrun=>ladc_overrun,
            dac_in_rdy=>ldac_in_rdy,
            dac_underrun=>ldac_underrun,
            sdin=>lsdin,
            sdout=>sdout
        );

    rchan_sel <= YES when lrsel=RIGHT else NO;
    rchan_on <= YES when lrck_int=RIGHT else NO;
    u_right: channel
        generic map
        (
            XSTEND_V1_2=>XSTEND_V1_2,
            DAC_WIDTH=>DAC_WIDTH,
            ADC_WIDTH=>ADC_WIDTH
        )
        port map
        (
```

```
                clk=>clk,
                sreset=>sreset,
                areset=>areset,
                chan_on=>rchan_on,
                bit_cntr=>bit_cntr,
                subcycle_cntr=>subcycle_cntr,
                chan_sel=>rchan_sel,
                rd=>rd,
                wr=>wr,
                adc_out=>radc_out,
                dac_in=>rdac_in,
                adc_out_rdy=>radc_out_rdy,
                adc_overrun=>radc_overrun,
                dac_in_rdy=>rdac_in_rdy,
                dac_underrun=>rdac_underrun,
                sdin=>rsdin,
                sdout=>sdout
            );

    dac_underrun <= YES when ldac_underrun=YES or rdac_underrun=YES else NO;
    adc_overrun <= YES when ladc_overrun=YES or radc_overrun=YES else NO;

    -- generates the serial data output to the SDIN pin of the
    -- codec DAC depending on which channel is being loaded
    sdin_int <= not(lsdin) when lrck_int=LEFT else not(rsdin);
    sdin <= not(sdin_int) when XSTEND_V1_2=false else sdin_int;

end codec_intfc_arch;
```