Connecting an I²S-Compatible Audio DAC to the AT91x40 Series Microcontrollers

Introduction

The purpose of this Application Note is to provide the procedure to construct the interface between a stereo audio digital-to-analog converter (DAC) and an AT91x40 Series Microcontroller (DocRef. 2).

The stereo audio DAC of choice for this example, is the Micronas® DAC 3550A device. The digital interface of such a device is usually I²S standard compliant. As the AT91x40 Series microcontrollers do not embed such a serial peripheral, a parallel to I²S serial interface has to be set up with a programmable logic device. This has been achieved with an ATMEL ATF1508ASV 128 Macrocells CPLD (DocRef. 3).

This class of DAC embeds audio configuration registers that are accessible through a serial link. Because this peripheral is not embedded in the AT91x40 Series Microcontrollers, the protocol must be established by software through Parallel Input/Output (PIO) lines (DocRef. 4).

References

The following sources contributed information to this Application Note

1. ATMEL AT91 ARM Thumb Microcontrollers AT91x40 Series Full Data-Sheet, Rev. 1354C-07/01
2. ATMEL High-performance EE PLD ATF1508ASV/ATF1508ASVL, Rev. 1408E-09/00
3. AT91 TWI Drivers for AT24C512 Serial E²PROM, Rev. 1742A-04/01
Serialization

The I²S (Inter-IC Sound) standard is based on a three-wire bus (DocRef. 1):

- a continuous serial clock (sck),
- a word select signal (ws)
- a serial data line (sd)

The device generating sck and ws (i.e., the CPLD in our application) is the master. The serial data (sd) is driven out from the master on the trailing edge of the serial clock (sck) and sampled by the audio DAC on the sck leading edge. The word select signal (ws) indicates the channel being transmitted:

- ws = 0; left channel,
- ws = 1; right channel.

The access to the serializer is established through the AT91 External Bus Interface (EBI) peripheral. An interrupt signal is provided by this device to the microcontroller in order to alert the software that it must write the next audio data.

To view the adopted I²S hardware configuration, please refer to the illustration in Figure 1 on page 3. The hardware connections are illustrated in Figure 3 on page 4. In this application example, the 6.144MHz Quartz Crystal Oscillator sets the audio data sample rate to 48KHz.
I²S Hardware Block Diagram

Figure 1. I²S Hardware Interface Block Diagram

Figure 2. Timing Diagram
Figure 3. I²S Hardware Connections

- AT91x40
- EBI
- CPLD ATF1508ASV
- 6.144 MHz Quartz Crystal Oscillator
- PIO
- TWI Bus
- i²S Stereo Audio DAC DAC 3550A
- i²S Bus
- Analog Outputs
The related CPLD VHDL Code is the following:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- Entity Section

entity i2s_interface is
port (
    data: in std_logic_vector(15 downto 0); -- Data Bus
    address: in std_logic_vector(3 downto 1); -- Address Bus
    nrst, -- AT91 Reset Signal
    ncs, -- AT91 Chip Select Signal
    cs, -- AT91 Chip Select Signal
    nwe, -- AT91 Write Control Signal
    clock : in std_logic; -- 6.144MHz Clock Input
    irq, -- AT91 Interrupt Signal
    dai, -- I2S Serial Data
    wsi, -- I2S Channel Word Select
    cli: out std_logic -- I2S Serial Clock
);
end i2s_interface;

-- I2S Architecture Section

architecture i2s of i2s_interface is

signal left_data_select, -- left data chip select signal
    right_data_select, -- right data chip select signal
    run_command_select, -- start/stop command chip select
    start, -- Start/Stop Serializing Data
    sck, -- primary clock (6.144MHz)
    clk_int_sck, -- intermediate clock variable
    sck_1, -- left data register output enable command
    sck_2, -- right data register output enable command
```
ws, wsp

: std_logic ;

signal left_data_1,
    right_data_1,
left_data_2,
right_data_2,
parallel_data,

loaded_data:
: std_logic_vector(15 downto 0);
signal clk_int_ws
: integer range 0 to 31;

begin

i2s_selection : process(nrst, cs, ncs, nwe, address)
begin
if ((nrst = '0') or ((cs = '0') and (ncs = '1')) or (nwe = '1')) then
    left_data_select <= '1' ; -- select signals are deactivated
    right_data_select <= '1' ; -- in case of reset assertion
    run_command_select <= '1' ; -- and no area selection
elsif ((cs = '1') and (ncs = '0') and (nwe = '0')) then
    case address is
    when b"000" =>
        left_data_select <= '0' ; -- if area selection,
        right_data_select <= '1' ; -- one select signal
        run_command_select <= '1' ; -- activated
    when b"001" =>
        left_data_select <= '1' ; -- depending on
        right_data_select <= '0' ; -- the address code
        run_command_select <= '1' ;
    when b"010" =>
        left_data_select <= '1' ;
        right_data_select <= '1' ;
        run_command_select <= '0' ;
    when others =>
        left_data_select <= '1' ;
        right_data_select <= '1' ;
        run_command_select <= '1' ;
    end case ;
else
    left_data_select <= '1' ; -- else, all select signals
    right_data_select <= '1' ; -- deactivated
    run_command_select <= '1' ;
end if ;
end process ;
left_data_1(15 downto 0) <= data(15 downto 0) when rising_edge(left_data_select) ;
-- left data registered (first registration stage) in case of related select signal activation

right_data_1(15 downto 0) <= data(15 downto 0) when rising_edge(right_data_select) ;
-- right data registered (first registration stage) in case of related select signal activation

start <= '0' when nrst = '0' else data(0) when rising_edge(run_command_select) ;
-- start/stop command registered in case of related select signal activation

clk_int_sck <= '0' when start = '0' else not clk_int_sck when rising_edge(clock) ;
sck <= '0' when start = '0' else not sck when rising_edge(clk_int_sck) ;
-- I2S Serial Clock

sck_div : process(nrst, start, sck)
begin
  if ((nrst = '0') or (start = '0')) then
    clk_int_ws <= 31 ;
  elsif falling_edge(sck) then -- clk_int_ws = sck
    clk_int_ws <= (clk_int_ws + 1) mod 32 ; -- divided 5 times
  end if ;
end process ;

ws_assert : process(nrst, start, clk_int_ws)
begin
  if ((nrst = '0') or (start = '0')) then
    ws <= '1' ;
  elsif ((clk_int_ws >= 0) and (clk_int_ws <= 15)) then
    ws <= '0' ; -- ws toggles each
  else -- sck divided 4 times toggling
    ws <= '1' ;
  end if ;
end process ;

-- I2S Channel Word Select

data_load_signals : process(start, ws, sck)
begin
  if (start = '0') then
    sck_1 <= '0' ;
    sck_2 <= '0' ;
  elsif rising_edge(sck) then
    sck_1 <= ws ;
    sck_2 <= sck_1 ;
  end if ;
end process ;
-- parallel data to be loaded into the shift register output enable signal
-- signal to generate the load of the shift register in combination with the sck_1 signal

wsp <= not (sck_1 xor sck_2) ; -- shift register load signal

registered_data : process(nrst, start, ws) -- data second registration stage
begin
  if ((nrst = '0') or (start = '0')) then
    left_data_2(15 downto 0) <= x"0000" ;
    right_data_2(15 downto 0) <= x"0000" ;
  elsif falling_edge(ws) then
    left_data_2(15 downto 0) <= left_data_1(15 downto 0) ;
    right_data_2(15 downto 0) <= right_data_1(15 downto 0) ;
  end if ;
end process ;
parallel_data(15 downto 0) <= left_data_2(15 downto 0) when sck_1 = '0' else right_data_2(15 downto 0) ;

-- registered data release to the shift register input

shifter : process
begin
  wait until falling_edge(sck) ;
  if (wsp = '0') then
    loaded_data(15 downto 0) <= parallel_data(15 downto 0) ;
    -- loading of the released data into the shift register
  elsif (wsp = '1') then
    loaded_data(15 downto 1) <= loaded_data(14 downto 0) ;
    loaded_data(0) <= '0' ;
    -- loaded data shifted with sck
  end if ;
end process ;

serial_data : process(nrst, start, loaded_data(15))
begin
  if ((nrst = '0') or (start = '0')) then
    dai <= '0' ;
  else
    dai <= loaded_data(15) ; -- output serial data
  end if ;
end process ;

cli <= sck ; -- I2S Serial Clock
wsi <= ws ; -- I2S Channel Word Select
irq <= ws ; -- AT91 Interrupt Signal

end i2s ;
Software Interface Description

The corresponding software interface is made up of 3 16-bit registers to write the right and left channel data and to start and stop the serialization process.

The I²S registers are accessed through an AT91 Chip Select line and the sub-addresses are decoded by the CPLD.

<table>
<thead>
<tr>
<th>Offset to Base Address</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Load input latch left channel</td>
</tr>
<tr>
<td>0x2</td>
<td>Load input latch right channel</td>
</tr>
<tr>
<td>0x4</td>
<td>Start (Write 1)/Stop (Write 0: Reset State)</td>
</tr>
</tbody>
</table>

Schematics

The schematics shown in Figure 4 on page 10 and Figure 5 on page 11, illustrate a concrete example of the principle set out in this Application Note.
Figure 4. Schematics Board: Audio Extension Card Decode CPLD
Figure 5. Audio Extension Card Audio DAC