High-End Audio Playback with the Parallella

1. Introduction

This project uses two techniques which complement each other to greatly improve the audio reproduction of standard CDs:

- Benefits of Bi-Amping from Rod Elliott
  http://sound.westhost.com/bi-amp.htm

- Digital Room Correction from Denis Sbragion
  http://drc-fir.sourceforge.net/

Tools needed

1. Daughter board with SPDIF in/out interfaces, ADC and high-precision clock (see below)
2. Measurement microphone (and possibly microphone pre-amp)
3. Crossworks for Arm toolchain and a Crossworks supported JTAG probe
4. CoolEdit (Adobe Audition) or similar PC utility
5. SPL (Sound Pressure Level) meter
6. Micro-SD card (i.e. standard 4GB)

Note: if the amplifier(s) do not have a SPDIF input, it will be necessary to add a DAC with SPDIF interface. I have not tried it but this one looks pretty good to me: http://www.beis.de/Elektronik/ADDA24QS/DA24QSDS.html

2. Hardware

A daughter board should be made to include:

1. Electrical interface for SPDIF input and output(s)
   (like the one found in Appendix at http://sound.westhost.com/project85.htm)

2. PCM3052 AD converter with high-precision clock
   Originally a separate module was built but it should be on the daughter board instead because both SPDIF input and output should use the same clock. Then instead of having a dedicated 8-bit MCU to set the ADC via I2C interface, it can be done directly with the Xilinx Zynq. The original module design files are available in the Audio Playback thread of the Parallella forum.

Note1: the Xilinx Zynq does not generate a lot of heat when running this application so the Parallella heatsink is sufficient (i.e. no fan needed)

Note2: this application does not use the Epiphany chip

3. FPGA

- copy "*HEAP_para7010_headless*" and "*ip_repo*" directories in Vivado projects sub-directory (i.e. C:\Users\pat\project2015_1)
- open "*HEAP.xpr*" with Vivado 2015.1
- set the proper constraints in "*PL_pins.xdc*" if necessary
- Generate Bitstream and Export Hardware

- open SDK 2015.1
- File - New - Project - Hardware Platform Specification:
  create "*curPL*" with bitstream (i.e. C:\Users\pat\project2015_1\HEAP_para7010_headless\HEAP.sdk\design_1_wrapper.hdf)
- Then generate the proper BSPs: File - New - Application Project
  create "*test*" (default with ps7_cortex9_0 and hello_world)
- build "*test*"

- Create Zynq Boot Image (import "*test.bif*" from the bootimage directory)
- Program Flash

Note: "*Program Flash*" will write the FSBL and the new FPGA bitstream in QSPI flash so that the FPGA will be programmed with that bitstream on each hard reset

Warning: "*Program Flash*" will erase the QSPI flash Adapteva factory code. To revert to the original Parallella setting, a "*Program Flash*" with the factory code will need to be performed

4. ARM

4.1. Measurements

<u>**CAUTION**</u>:
Use extreme care and make sure you understand "*DRCGuidev1.0.pdf*" well before going ahead
*DRCGuidev1.0.pdf is a guide written by Jones Rush that describes the DRC setup when using a PC-based audio playback device instead of the Parallella one

Note1: during measurements I preferred to split some operations in order to closely monitor and double-check the intermediate results, but it should be pretty straightforward to wrap more or less everything in one go

Note2: there is no user interface. Typically you need to set parameters in the main.c audioTask() function (type/track/bypassMeasure), build the application and then execute it to perform a given task

Note3: both the target java script "*Zynq_7000_Target.js*" and the Zynq-PS initialization file "*init_ps7_7010.bin*" need to be accessed from an absolute reference path from Crossworks, so you must first set to your own path:

- In Project Properties – Target Script File for "*Zynq_7000_Target.js*"
- In function DDRReset() of "*Zynq_7000_Target.js*" for "*init_ps7_7010.bin*"

- open "*cdplayer_7010.hzp*" solution with Crossworks
- for bi-amping add "*XOVER*" to Preprocessor definitions (in Project Properties)
- for room correction add "*DRC*" to Preprocessor definitions (in Project Properties)
- in main.c function: audioTask() set:
  SourceType type = LOGSWEEP_SIN;   // WAVEFILE/CDPLAYER/LOGSWEEP_SIN
  SourceTrack track = STEREO_LEFT; // STEREO_LEFT/STEREO_RIGHT (LOGSWEEP_SIN only)
  int bypassMeasure = FALSE;   // if TRUE do LOGSWEEP_SIN IR conversion only (testing)
- if using bi-amping, set the "cutoff1" parameter in main.c function: paramsInit() (currently set to 200Hz)
- build the solution
- connect a cable to loopback SPDIF output to input
- run the application (JTAG probe connected to the Parallella)
- the message "*audioTask status = 0*" should be displayed when the cycle is completed after about one minute
- Insert the SD card in a pc and check the "*MSLEFT.WAV*" file with a utility like CoolEdit (blank/trigger/logsweep/blank)

Note: the loopback test will <u>only</u> work when *SourceTrack track = STEREO_LEFT;*

Warning: when running measurements in <u>THUMB Debug</u>, the program may hit the "*undef_handler*" because the bandwidth is a little too short

<u>Volume level adjustments</u>

<u>CAUTION</u>:

Setting the proper volume level is definitely the trickiest part of the whole procedure: too low and the SNR (Signal Noise Ratio) will not be good enough, too high and the tweeter will blow.

Tweeters are delicate piece of equipment and there are at least 3 ways to destroy one:

- Apply low frequencies (i.e. below the tweeter frequency range)

- Apply a volume level too high
- Apply a DC component

Please refer to the "*DRCGuidev1.0.pdf*" for setting guidelines.

- Starting with a low level volume, output a logsweep and measure the SPL at the listening position (tweeters are pretty directional and should be positioned roughly at the listener's ears height)
- Increase the level for each logsweep sequence until the SPL meter returns a <u>maximum</u> of **85dB at peaks**
- Adjust the microphone pre-amp in order to get the maximum dynamic range without clipping
- Adjust each amplifier volume level so that the SPL is roughly identical for all speakers

<u>Delay adjustments</u> (for bi-amping)

- For respectively *SourceTrack track = STEREO_LEFT;* and *STEREO_RIGHT*
- Measure the logsweep on each individual speaker (turn off irrelevant amps)
- Convolve with inverse logsweep to get the impulse response (*int bypassMeasure = TRUE;*)
- With CoolEdit note the sample position of the first pulse highest value
  - the first pulse, NOT the biggest one (also it could be negative)
  - eventually confirm by measuring the distance between speakers and calculating it:
    1 second -> 44100 samples -> 343 meters
- set delays accordingly in main.c: initDelay() in order to time-align each speakers to the slowest one

Note: speakers time-alignment is important and should be done very carefully

<u>DRC</u>

After full speaker measurements in MSLEFT.wav and MSRIGHT.wav:

- with CoolEdit locate trigger recorded peak after trigger sample 0xa55a
  (eventually confirm by checking 1024*1024*2 samples after and look for another small peak)
  In my case, the speakers being roughly 2m from the listening position, the trigger peak was 825 samples after the trigger 0xa55a (corresponds to the recorded sound generated by the trigger)
- set bypassMeasure to TRUE
- build and run in **THUMB Debug**
- in ir(char *filename, int nSamples) (ir.cpp) insert a breakpoint at "*if ((pos = locateSinewave(...*"

- after executing the function, manually set "*pos*" to the value + 1 previously found with CoolEdit (*i.e. 826 in my case*)
- resume program execution until completion (i.e. "*audioTask status = 0*")
- do the same for both track = STEREO_LEFT and STEREO_RIGHT
- copy IRLEFT.pcm and IRRIGHT.pcm to PC
- download and unzip drc-3.2.1 in PC
- in D:\Audio\drc-3.2.1\sample edit "*normal-44.1.drc*" to point to: "*BCInFile = irleft.pcm*"
- run "*drc normal-44.1.drc*" and after completion rename the created "*rps.pcm*" to "*drcLeft.pcm*"
- do the same with IRRIGHT.pcm
- with CoolEdit create a "*drc.wav*" file for which left track is "*drcLeft.pcm*" and right track is "*drcRight.pcm*"
- copy "*drc.wav*" to the SD card root directory
- generate 10s of stereo white noise, normalized to 100%, copy drc.wav to clipboard and Aurora convolve white noise with clipboard (see details in "*DRCGuidev1.0.pdf*")
- use resulting value to set convolverAdjust in paramsInit() (in main.c)

- change to "*SourceType type = CDPLAYER;*"
- rebuild cdplayer_7010 in **THUMB Release**
- copy resulting "*cdplayer_7010.elf*" into the bootimage directory

in SDK 2015.1

- Create Zynq Boot Image (import "*cdplayer_para.bif*" from bootimage directory)
- Program Flash

Connect a CD player to the SPDIF input, sit back and enjoy the music!