Project Design Document

MANIC (MP3 Player)

Prakash GS(pg2132),
Vijayaraka N(vn2107),
Devyani Gupta(dg2168)

1. Introduction:

We propose to build an mp3 player, using the existing onboard sound processing peripherals.

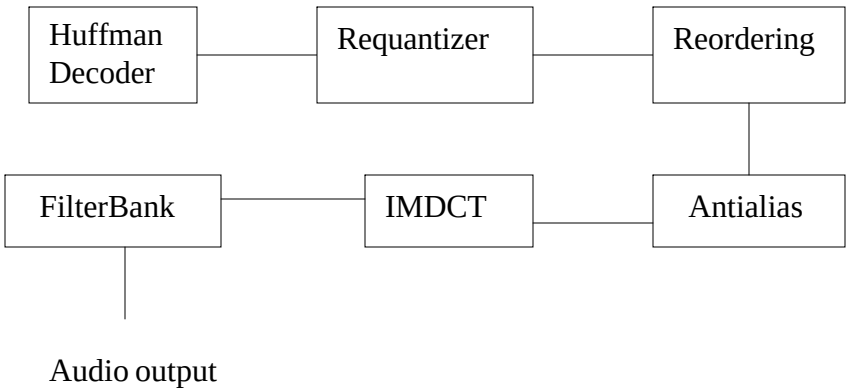
2. MP3 Decoding Algorithm:

We intend to use MAD (Mpeg Audio Decoder) library for mp3 decoding. MAD is available under the terms of the GNU General Public License .There are several advantages of using MAD:

- 24-bit PCM output
- 100% fixed-point (integer) computation

Performing floating-point arithmetic delays the decoding process. Hence the MAD decoder was chosen.

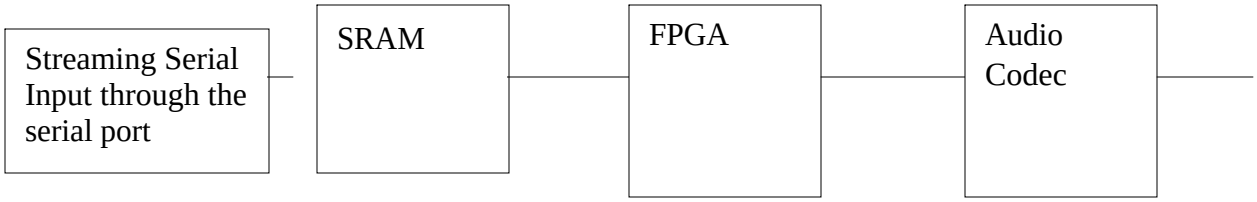
The decoding process basically consists of the following parts:



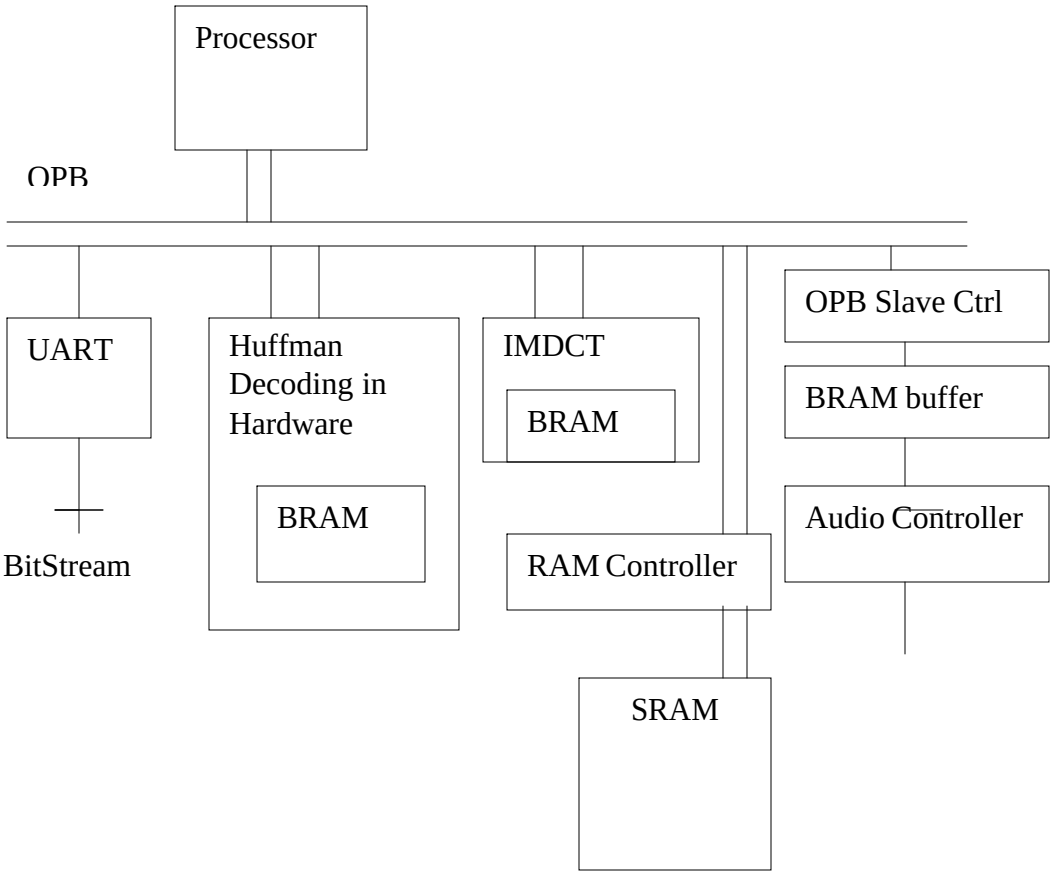
3. Implementation:

Since a complete software implementation of the algorithm would be slow, we have decided to implement a few parts of the algorithm in VHDL. After an analysis of some opensource decoders like mpg321 on a standalone Linux box using ***gprof,*** we have observed that the functions implementing the **Huffman Decoding and the IMDCT** took the most amount of time. Hence we have decided to implement these two processes in VHDL.

4. Control Flow:



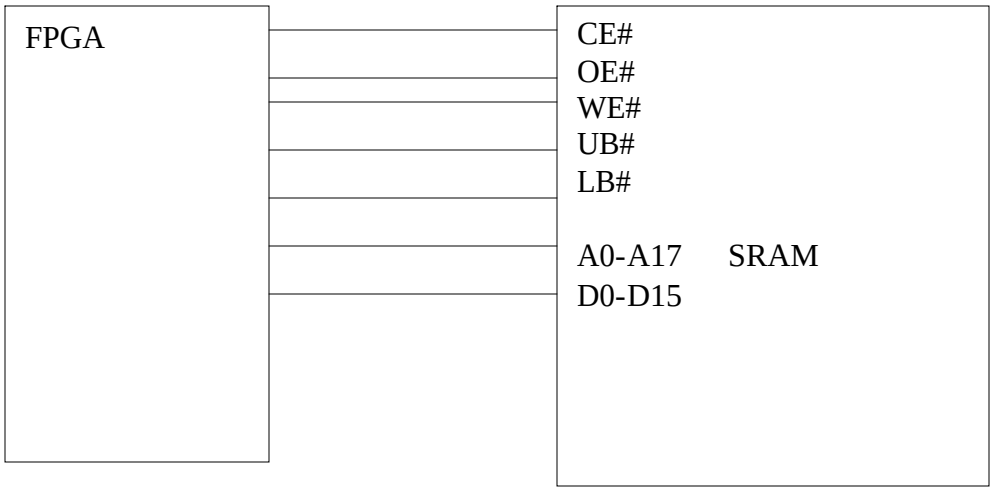
Block Diagram



The control flow can be broken down into the following:

1. Reading mp3 from a host via the Serial port/ Ethernet controller:

Depending on the output quality of the sound (i.e. ranging from 32 kbps to 128kbps) we have decided to use the serial port or the Ethernet to stream mp3 data to the SRAM. Interrupts are used to manage switching between the two tasks of reading mp3 data from the SRAM and decoding and streaming data into the audio device. The Ram Controller will be used to generate the following signals to the SRAM.



2. Decoding the mp3 :

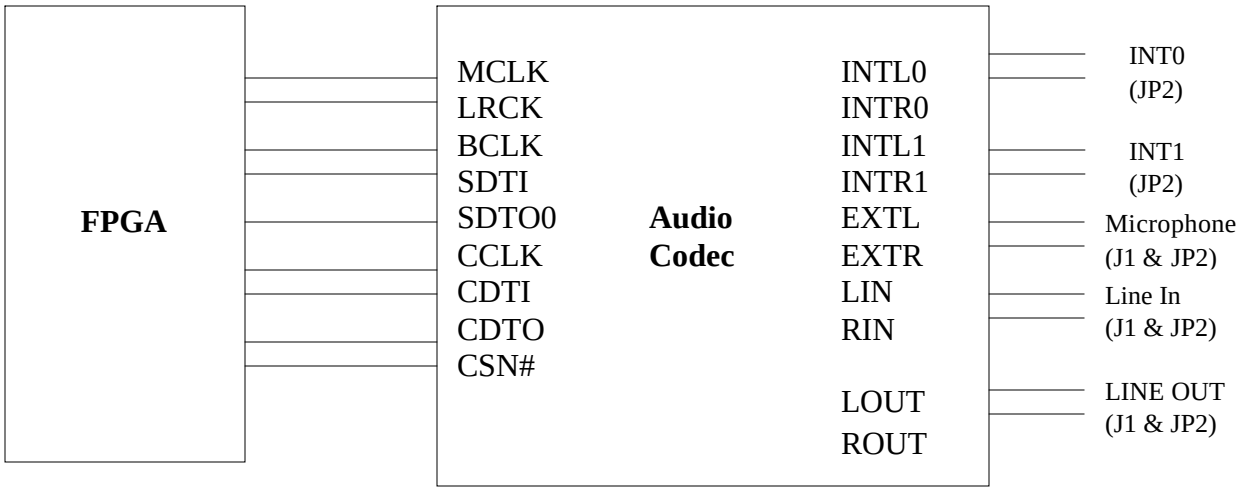
The following steps are implemented in the MP3 decoding algorithm.

a) Huffman Decoding (VHDL Implementation)
b) Requantizer
c) Reordering
d) Antialias
e) IMDCT(VHDL Implementation)
f) Filter Bank

Huffman Deocding and IMDCT will be implemented in hardware. We will use the available BRAMs on the FPGA, to communicate to the Huffman core as well as the IMDCT core as shown in the block diagram. The remaining processing (requantizer, reordering, antialias and filter bank) will be done in software (C programming) on the Microblaze processor. The main loop of the microblaze would be processing these steps, waiting on the results from the Huffman core and the IMDCT core. The interrupt handler will be used to stream in data from the serial port and store it on SRAM.

3. Sending decoded audio to the audio codec (AK4565):

The FPGA streams the decoded bit stream serial (through the SDTI pin) to the audio codec through the BRAM buffer. The buffer can store 4 kilobits of data. Each sample is of 32 bits. So at a time, we can store upto 128 samples on it. If our sampling frequency is 44.1 Khz, 128 samples would require 3 ms. The audio controller would generate an interrupt after every 1.5 ms, indicating that the buffer is half full. From the buffer, the serial bit streams are synchronized with a clock from the FPGA that enters the codec on the BLCK signal. The master clock from the FPGA (MCLK) synchronizes all the internal operations of the codec. The FPGA uses the LRCK (Left Right Clock) to select the left or the right channel as the destination of the serial data.



s

The following signals to the Codec from the FPGA are generated from the audio controller.

1. MCLK
2. LRCK
3. BCLK
4. CSN#

Data is passed in through SDTI

The analog streams are passed to LOUT and ROUT.