

# MP3 Player

**4840 Final Project Presentation**

**-Zheng Lai -Zhao Liu**

**-Quan Yuan -Meng Li**



# Motivation

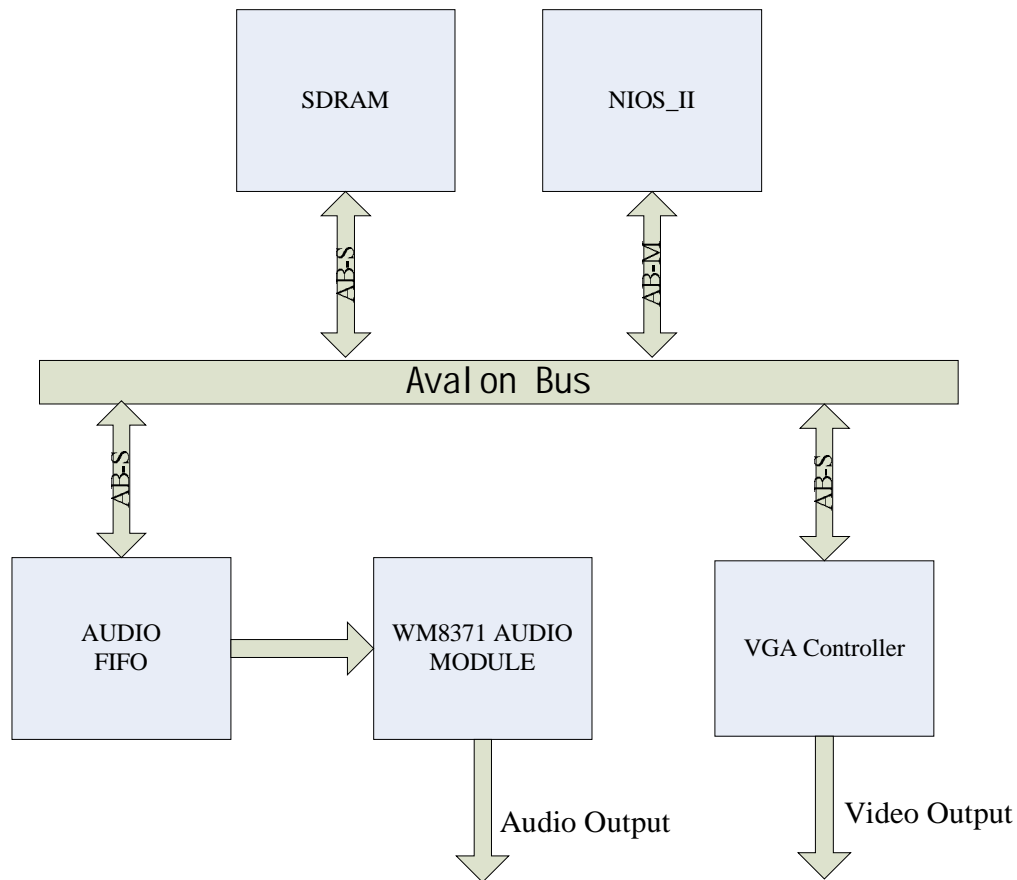
- **We originally want to design a game that are similar to Guitar Hero, combining MP3 player, keyboard control and video game. But find out that the plan is too big.**
- **Then we designed to do MP3 Player with frequency display on the board**



# Project Overview

- **Implemented software MP3 decoder**
- **To the raise the speed of decoding, optimized original decoding algorithm**
- **Play out the decoded music**
- **Display frequency information as histogram on the screen**

# Hardware Architecture

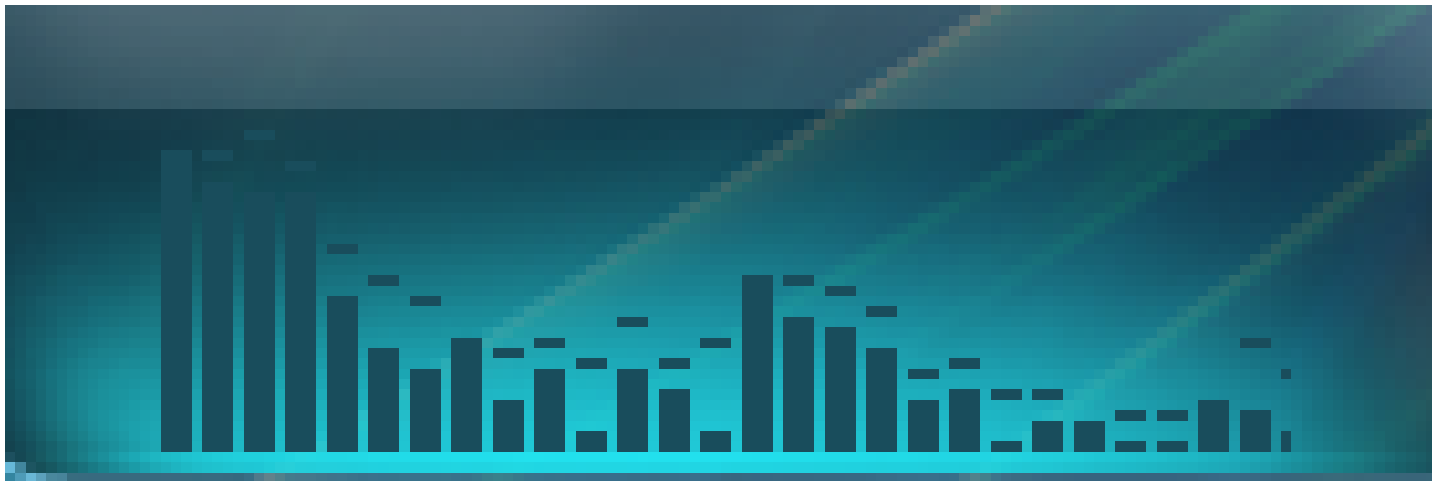




## VGA output control module

- **Since we want to display the amplitude of signals of 64 different frequencies, a histogram with 64 rectangles are shown on the screen first.**

- The vertical starting points of each rectangle are defined with frequency information from the output of the frequency analyzer. In this way, frequency information is shown by rectangle shapes. The whole histogram looks like those on the classic media players.

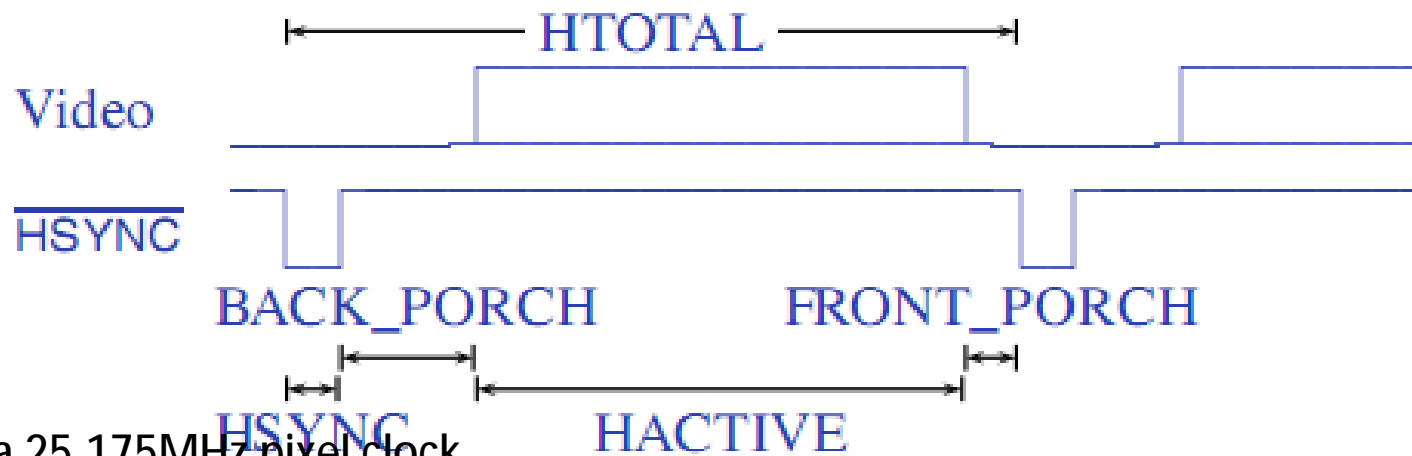




1. Show histogram formed with 64 rectangles.

- **First, to show 64 rectangles on screen, the horizontal starting points, horizontal ending points and vertical ending points of each rectangle are fixed.**

The following figure shows the horizontal timing of the VGA: (vertical timing is similar)



For a 25.175MHz pixel clock,  
HSYNC=96 pixels  
HBACK\_PORCH=48 pixels  
HACTIVE = 640 pixels  
FRONT\_PORCH=16 pixels  
HTOTAL =800 pixels



- Define a signal of Hcount. At every rising edge of clk25, Hcount add 1, therefore, Hcount works as a horizontal pointer of the scanning position.
- When  $Hcount = HSYNC + HBACK\_PORCH + RECTANGLE\_HSTART$ , where RECTANGLE\_HSTART represents the horizontal starting point of the rectangle, it means the pointer is moving into the horizontal area of rectangles in the histogram.
- When  $Hcount = HSYNC + HBACK\_PORCH + RECTANGLE\_HEND$ , where RECTANGLE\_HEND represents the horizontal ending point of the rectangle, it means the pointer is moving out of the horizontal area of rectangles in the histogram. During the clock cycle and the clock cycles following, the area on the screen will **black out**.

- Having the same function with Hcount, signal Vcount works as a vertical pointer of the scanning position.
- When  $Vcount = VSYNC + VBACK\_PORCH + RECTANGLE\_VSTART$ , where  $RECTANGLE\_VSTART$  represents the vertical starting point of the rectangle, it means the pointer is moving into the vertical area of rectangles in the histogram.
- When  $Vcount = VSYNC + CBACK\_PORCH + RECTANGLE\_VEND$ , where  $RECTANGLE\_VEND$  represents the vertical ending point of the rectangle, it means the pointer is moving out of the vertical area of rectangles in the histogram. During the clock cycle and the clock cycles following, the area on the screen will **black out**.


- The necessary condition for a point on screen to **light up** is :

$Hcount = HSYNC + HBACK\_PORCH + RECTANGLE\_HSTART$

$Vcount = VSYNC + VBACK\_PORCH + RECTANGLE\_VSTART$

This means the pointer begins to enter into both the vertical area and horizontal area of rectangles.

- The 64 rectangles in the histogram are all realized in this way.



2. Control the height of each rectangle with the frequency information from the output of the frequency analyzer.

- **The 64 vertical starting points of the 64 rectangles are determined with frequency information from the output of the frequency analyzer. In this way, frequency information is shown by heights of each rectangle.**
- **According to the situation that the active area of the screen is  $640 \times 480$ , the ending point of each rectangle in the histogram is chosen to be 380, therefore, the frequency information input should be from 0 to 380.**



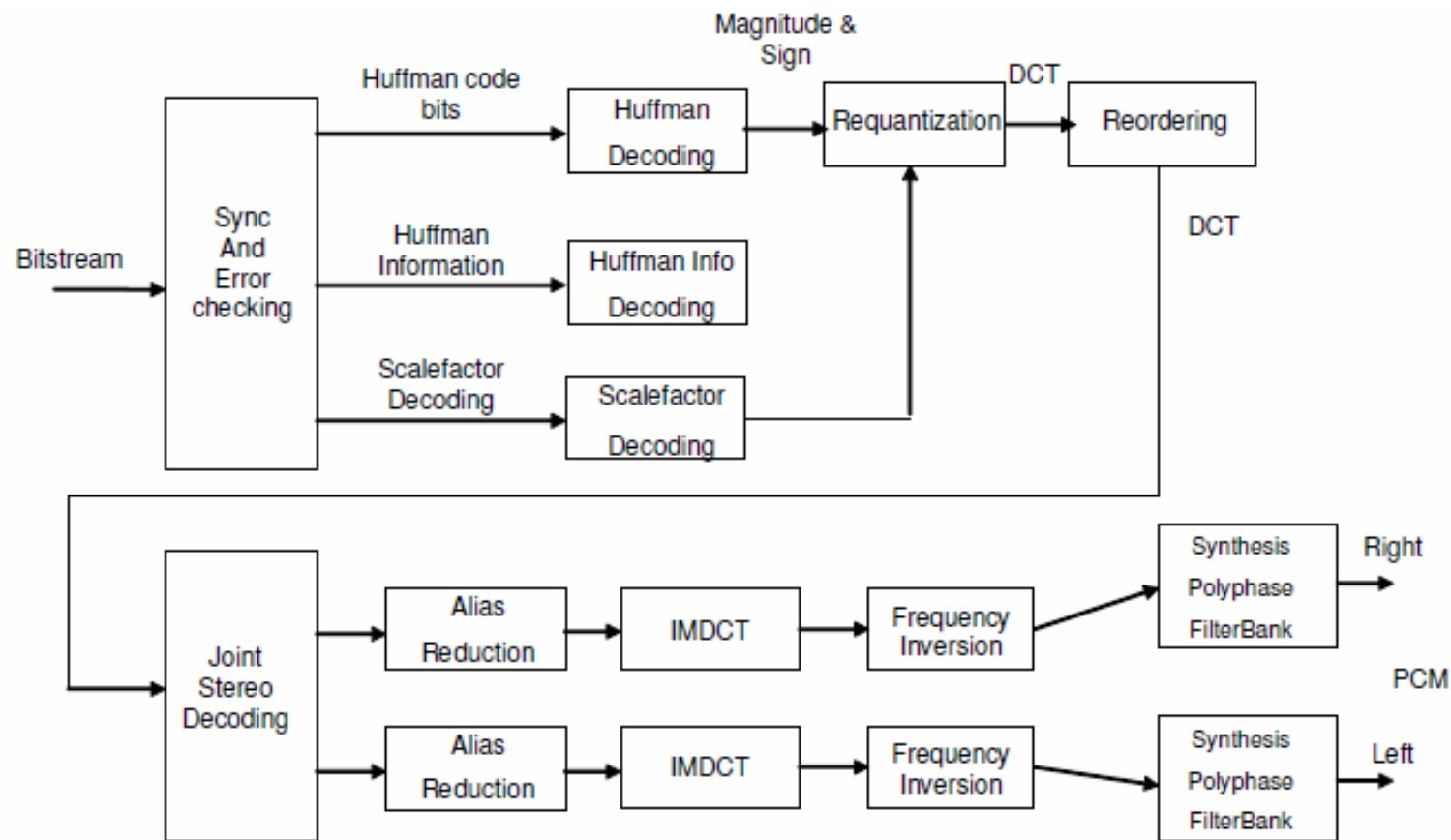
## MP3 Decoding (Software)

- **When doing each part, compare the result of open source code and ours to verify the correctness**
- **The very first parts that need to work on input(file or data array) was actually harder since more bugs occurred when we were doing these parts**

# MP3 Decoding (Software)

- **Problem:**
- **We don't have an operating system. It's difficult to transplant an open-core MP3 decoding library on DE2 board**
- **We don't have a file system either**
  
- **We translate the bit-stream in MP3 file to a huge array. All decoding processes are operating based on this array**
- **We study the *ISO/IEC 11172-3* carefully and raise a program based on the algorithm proposed in the ISO document. Before milestone #2, all time is spent on making the decoder program on NIOS II to output a same result compared with that on a desktop PC.**

# MP3 Decoding (Software)





# Floating Histogram

- **How to extract the frequency information for one frame? Luckily, inputs of the function IMDCT (inversed modified discrete cosine transform) contains frequency information for a frame.**

**We take these frequency information and use it to control the heights of 64 rectangle to exhibit a floating histogram effect.**

**( The floating histogram may not reflect the accurate frequency information. Since it should be overlaid with its next and previous sub-band. But we only need a visual effect)**





# Floating Histogram

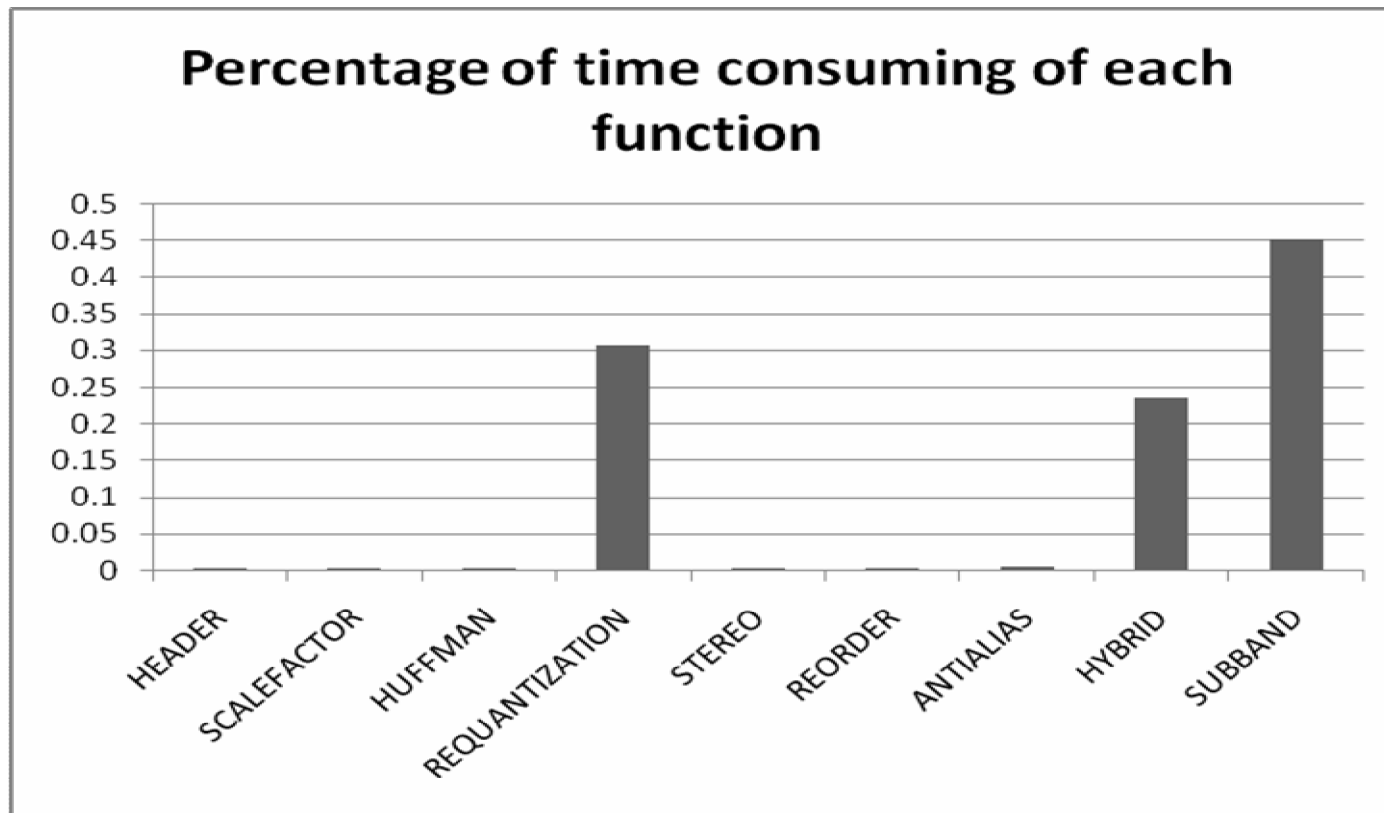
- **The original frequency information are single precision variables. How to quantize?**

**We use low level programming , translate these variables to fixed point format. After that the translation result can be applied to the VGA controller with minor computation. This speeds up the decoding process, too.**

# Optimization

- **MP3 decoding in PC is fast, but in the DE2 board is slow!**
- **A piece of 3-minute MP3 music has thousands of frames**
- **Original program: decoding one frame needs more than 20 minutes! (using Nios II/e processor)**
- **Even with Nios II/f, decoding one frame still needs 5 minutes**

# Optimization



**\*IMDCT is in the “HYBRID” function**



# Optimization

- **So we optimize algorithm**
- **Attempt1: Change the mathematical function to lookup table**
- **Attempt2: Custom floating point operation function**
- **Attempt3: Change floating point precision, from double to single**

# Optimization #1

- **Look up table**

**For example, in the requantization process,**

$$xr_i = \text{sign}(is_i) * |is_i| \left| \frac{4}{3} * 2^{\frac{1}{4}} (\text{global\_gain}[gr] - 210 - 8 * \text{subblock\_gain}[\text{window}][gr]) \right. \\ \left. * 2^{-(\text{scalefac\_multiplier} * \text{scalefac\_s}[gr][ch][sfb][\text{window}])} \right.$$

**The combinations of those integer variables (such as global gain, sub-block gain ) are limited. We can pre-calculate [xr] for each combination of variables and store them in a big look-up tables.**

**We also generate look up tables for Sin and Cos in the process of hybrid synthesis.**

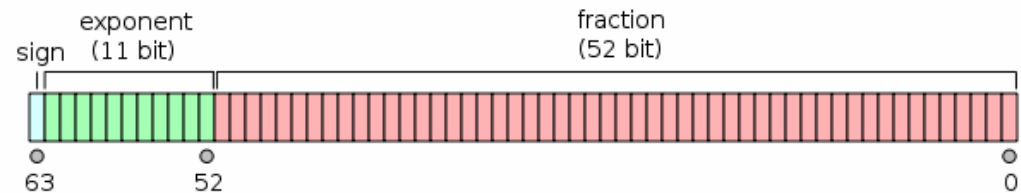


## Optimization #2

- **Revised algorithms for floating point computation (Not using standard C library). Our goal is speeding up the decoding process without loosing accuracy.**
- **The resulting algorithms contains only shifting operation and integer multiplication.**

**Yield at least 16 bits of precision in the “fraction” Part.**

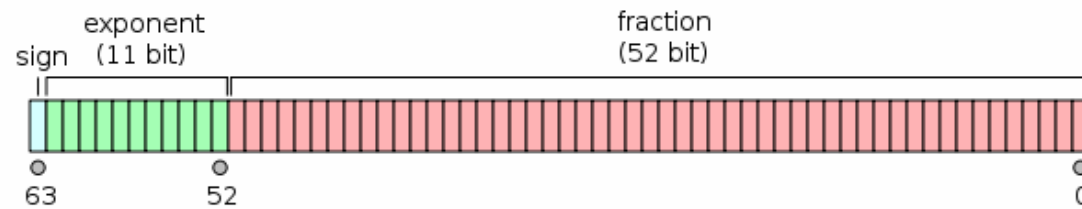
# Optimization #2



- **Floating Point Multiplication**
- **Pseudo Code for double precision multiplication:**
- **Double x, Double y;**
- **If x==0 || y==0;**
- **Return 0.0;**
- **Else**
- **Bit operation to extract sign,exponent,fraction;**
- **fraction|=0x0010000000000000;**
- **temp=(fraction\_x>>27)\*(fraction\_y>>27); // integer multiplication**
- **test=(0x800000000000000&temp)>>51; // test the position of first 1;**
- **If test==1{**
- **fraction\_result=(0x7FFFFFFFFFFFFFFF&temp)<<1;**
- **exp\_result=exp\_x+exp\_y-1022;**
- **}**
- **Else {**
- **fraction\_result=(0x3FFFFFFFFFFFFFFF&temp)<<2;**
- **exp\_result=exp\_x+exp\_y-1023;**
- **}**
- **Return outcome=(sign\_result<<63) | (exp\_result<<52) | fraction\_result;**

# Optimization #2

- **Floating Point Addition**



- **Pseudo-code is too long and not shown here. But the principle is the same. First extract the fraction and exponent. Then do integer multiplication and shifting accordingly.**





## Optimization #3

**The Revised algorithms are still too slow! With look-up table and these algorithms decoding for one frame roughly takes half a minute. But one frames last 26ms long.**

- 1) We switch from double precision to single precision.**
- 2) Implement a custom instruction dedicated to compute single precision floating point**

# Optimization

- **Optimization Result.**
- **We can decode one frame within roughly 0.6 sec! Remember without any optimization it takes roughly half an hour!**

	Decoding time (s) per frame (timed by timer)	Decoding time (s) per frame (timed by watch)
Initial implement	-	1560
Un-optimized	46.7	300
Software Optz.	12.8	30
Hardware Optz.	0.58	1



Finally..

**However,**

**1. We can not still meet the speed requirement.**

**Still need at least 20 times faster(26ms)**

**2. There are noise and distortion coming out from WM8731. (The decoded PCM samples can be played on DELL PC correctly, without noise)**

**Maybe we should not emphasize too much on accuracy, and switch to fixed point computation.**



Thank you