Niklas Blomqvist, Philip Johansson, Matteus Laurent, Johan Levinsson, Oscar Petersson, Erik Peyronson

Group 41 - Design Specification

Högskoleingenjörsutbildning i datateknik, 180 hp

Design Specification - October 8, 2015
System Design - Project, HT15
TSIU03, Linköpings universitet

Supervisor:
Petter Källström
Department of Electrical Engineering (ISY)

Contents

1. Introduction

Project 41 is based around audio signal processing. The audio input and output both go through the WM8731 chip on a DE2 board. Meanwhile, the hardware settings are controlled from a PS/2 keyboard and displayed on a VGA screen. The hardware settings to be implemented are a volume control and a balance control. In addition, an interface consisting of the input and output power level along with appropriate indicators as stated in the requirement specification.

The WM8731 is a stereo codec, which in Project 41 is used as a bridge between the audio source and a class-D amplifier. The custom hardware controls the WM8731 as the analysis of the input controls and encoding the graphical output. The output sound sent to a Class-D amplifier is then allowed further amplification through another instance of pulse width modulation within the amplifier.

2. System Level Description

This chapter will describe the system main blocks, the functionality of each of them, and the interaction between each block and its adjacent modules. Presented below (Figure 2.1) is a graphical overview of the system and its first layer of modules.

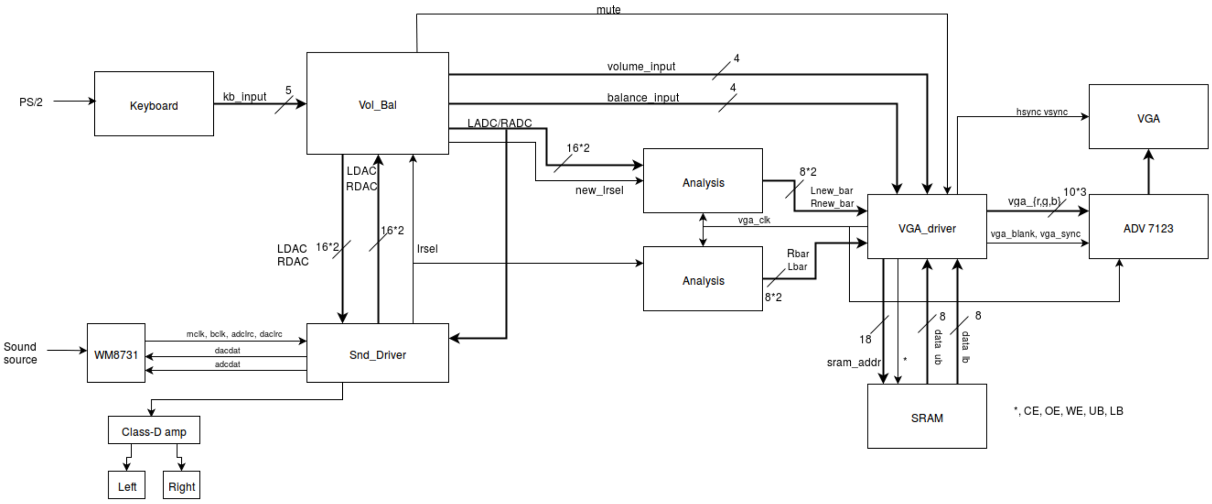

Figure 2.1: A graphical overview of the system's first module layer. Letters inside curly braces indicates multiple signals exlusively including each of the letters.

2.1 Keyboard

The user interacts with the system through a PS/2-connected keyboard. The keyboard is then handled by the module `Keyboard` which reads the scan codes, matches these against a *one hot encoded* preset which makes up the `kb_input` signal passed to `Vol_Bal`.

The module inputs are `PS2_DAT`, `PS2_CLK`, `clk` and `rstn` which are used to shift in the scan code and compare the result with the preset, resulting in `kb_input` — a 5-bit unsigned value indicating if either of the arrow keys have been released. `Vol_Bal` will then use this signal to adjust the volume and balance level. The Up/Down arrow keys controls the volume, and the Left/Right arrow keys controls the stereo channel balance.

The scan codes for the arrow keys consists of two (make code) or three (break code) bytes of information. These codes correspond to each other in the manner listed in figure 2.2.

The scan codes are shifted into a 26-bit shift register which is reset to all ones, and once the start bit (0) is shifted out, the third byte (bit 23 ... 16) is NAND:ed with FF_{16}. A result of "1" then compared to the expected third byte of a released control key which on success sends a `kb_input` to `Vol_Bal`.

*Figure 2.2: PS/2 Scan Codes used, corresponding **kb_input**, and how the input affects the system.*

KEY	MAKE	BREAK	`kb_input`	Function
U ARROW	E0,75	E0,F0,75	00001	Volume Increase
L ARROW	E0,6B	E0,F0,6B	00010	Balance Bias Left
D ARROW	E0,72	E0,F0,72	00100	Volume Decrease
R ARROW	E0,74	E0,F0,74	01000	Balance Bias Right
END	E0,69	E0,F0,69	10000	Mute Volume

2.2 Snd_Driver

The `Snd_Driver` module is an audio signal coder/decoder. It translates the signal between a parallel format and a bit serial format. The parallel format is sent to the `Vol_Bal` module which processes the sound and sends it back. The bit serial format is used by the WM8731 chip. This module will be a complete copy of the module used in `Laboration4`. In this case the module `Vol_Bal` will replace the `Application` used in that laboration.

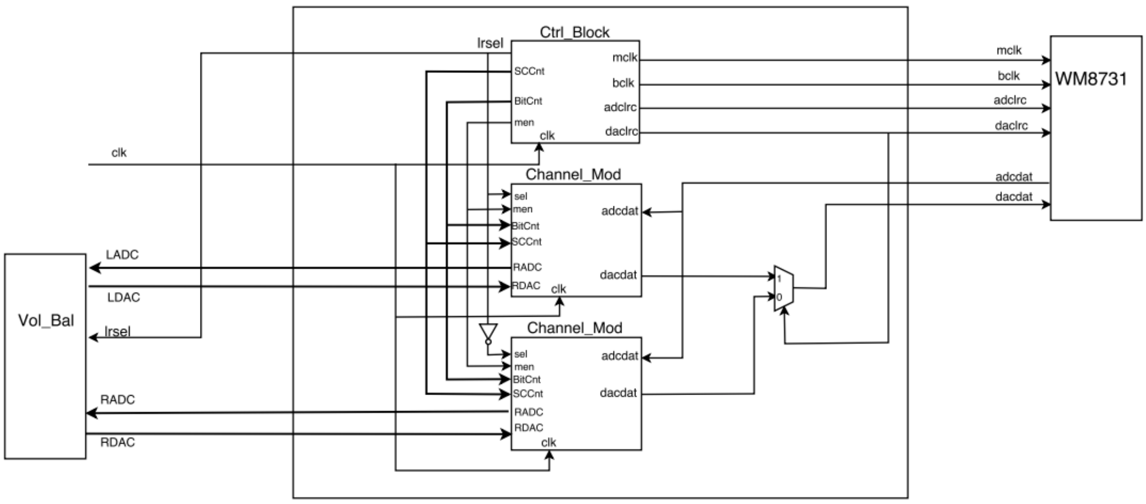

Figure 2.3: The Snd_Drivers scheme, as seen in Lab 4, Audio Codec.

2.3 Vol_Bal

The Volume/Balance module (`Vol_Bal`) acts as the hub for processing incoming digital audio signals, forwarded from WM8731 via the `Snd_Driver` module. As such, `Vol_Bal` also keeps internal registers that holds current volume and balance levels as signed 4-bit values (legal values range from -5 to 5 where 0 represents no adjustment). These registers update via the one-hot coded input signal `kb_input` applied by the `Keyboard` module. Consequently, the values they hold are not only used as signals (`i_volume_lvl`, `i_balance_lvl`) for the internal submodules that process the `LADC` and `RADC` inputs, but also as module outputs connected to the `VGA_Driver` so that they can be rendered on the screen.

The main function of the Volume/Balance module is to make requested adjustments to incoming values `LADC` and `RADC`, which represent measured amplitudes of the sound signal at distinct times. They will first be adjusted for volume by a function $A_{new} = A_{old} * \sqrt{2}^n$, where A is the amplitude and n is the signed value in the volume level register. The new values are forwarded for balance adjustment jointly with a `ready` signal to inform that the `adj_LADC` (or `RADC`) should be read. Same processing is applied in

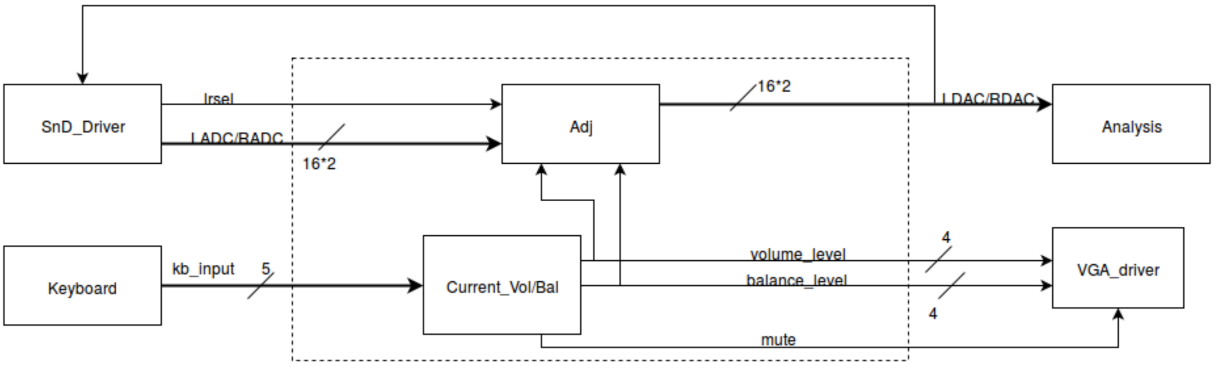

Figure 2.4: An overview of the Vol_Bal module's internal workings.

the `Bal_Adj` submodule to produce the `LDAC` and `RDAC` outputs conveyed to `Snd_Driver` and `Analysis`. Both adjustment submodules also use the input `lrsel` as a control signal.

Ultimately, the user have the ability to digitally adjust the volume by -15/+15 dB and additionally balance it by decreasing volume by up to another 15 dB on a single left/right audio channel. There is also a mute function which is conveyed by `kb_input`. When active, the register used as a "mute enable" essentially blanks any A_{new} values on the `LDAC/RDAC` outputs.

2.4 Analysis

The `Analysis` module has one responsibility. It reads the ADC signal and puts out information on how to draw two bars (one for each speaker) that represent the amplitude of said signal. Since we want bars for before and after modulation, we'll use two instances of the same module.

The required inputs include a left or right selection signal to specify which stereo channel we're about to analyse, two ADC signals (left and right), a clock signal that's synced with the `VGA_Driver` since the polling rate of the `Snd_Driver` and `VGA_Driver` differ.

There are two output signals (once again, one left, one right). They determine the height of the bar which the `VGA_Driver` should render.

`lrsel` determines which stereo channel should be read and thus which bar height should be written to at any given time.

2.5 VGA_Driver

The `VGA_Driver` module exists to handle the rendering of a 640x480 resolution image and the bar-graphs on the VGA display. The image being rendered consists of a background image previously stored in the SRAM consisting of prefilled bars that within the module will be blanked out according to the input stimuli, which will give the appearance of bars being filled to different levels.

To render an image on the VGA screen, five main signals is needed. Three analog color channels (red, green and blue) and two signals for synchronization `hsync` and `vsync`. The image is rendered pixel by pixel line by line using a horizontal sweep pattern which is reset by the two sync signals. If a color is set when the sweep resets arbitrary patterns can occur and therefore the signal has to be blanked during the reset phase.

The module `VGA_drive` has four input signals and five output signals described in table 2.6. It consists of eleven sub modules which can be over viewed in 2.5 described below.

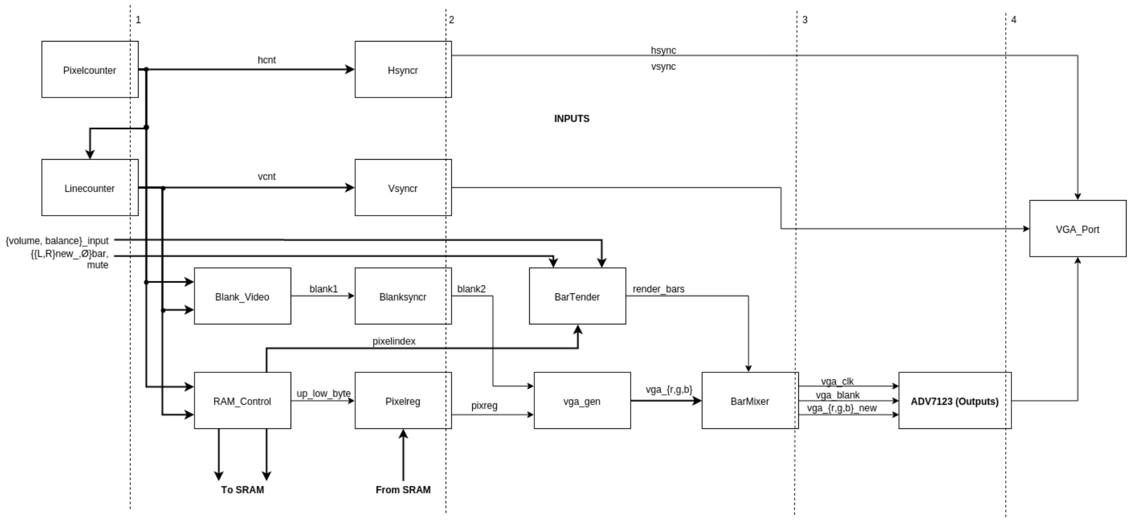

Figure 2.5: Block diagram of `VGA_Driver`

Figure 2.6: List of input and output signals

Name	Type	Description
`volume_input`	Input	a A bit input containing volume information
`balance_input`	Input	A 4 bit input containing balance information
`bar`	Input	A n bit input containing signal sound input signal level
`new_bar`	Input	A n bit input containing manipulated input signal level
`VGA_clk`	Output	Clock signal needed for scanning
`VGA_blank`	Output	A blanking signal for blanking when resetting scan
`VGA_(r,g,b)`	Output	Three signals containing color information

2.5.1 VGA_Driver:Pipelining

Since there is a time delay for the system to calculate ram address, generate `pixel_index` the system is pipelined. The four pipe-line stages are illustrated with dashed lines in figure 2.5. This means that there is a time delay of 3 clock cycles between which pixel is handled and which is drawn.

2.5.2 VGA_Driver:Linecounter

`Linecounter` works almost the same as `Pixelcounter` and generates the signal `vcount`. `vcount` is incremented with one every time `hcount` resets and resets after reaching 525. This makes `hcount` and `vcount` together act as coordinates to each pixel on the screen during the scanning.

2.5.3 VGA_Driver:Clock_Divider

`Clock_Divider` is just as its name suggests a clock divider which divides the system clock of 50 MHz to 25 MHz. This is needed because the timings in the VGA-interface for the resolution used requires a clock of 25 MHz.

2.5.4 VGA_Driver:Blank_Video

Due to the nature of the sweep in the VGA-interface the signal needs to be blanked just before, during and after the synchronization signals. This is done using the `Blank_Video` sub module, the 1 bit blanking signal is active low and should therefore be zero while outside of the visible image and and one inside. This is done by setting `blank` high when `hsync` is between 0 and 639 and `vsync` is between 0 and 479.

2.5.5 VGA_Driver:RAM_Control

This module is identic to the `RAM_Control` module in the third laboration with the exception of `pixelindex`, which is a counter for each pixel in the visible area. It is used by `BarTender` (see section 2.5.6

2.5.6 VGA_Driver:BarTender and BarMixer

`BarTender` is the submodule responsible for rendering the bar graphs displaying volume, balance and signal strength before and after signal manipulation. The background image already has the bars drawed filled and to give the appearance of them being filled to different levels pixels will be blanked out from the top down. Using `volume, balance, bar, new_bar` and `pixelindex`, `BarTender` will calculate which pixels should be blanked and set the signal `render_bar` high.

`BarMixer` works as a multiplexer blanking out the bars. The color information is passed through if `render_bar` signal is low and blanks out the pixel if high, which gives the effect of bar graphs being filled.

3. Challenges in the Design and Proposed Approach

The major problem in the project is the merging of the different modules together at top level. Most of the modules will be written and debugged separately and only need minor adjustments before they have been used in a bigger system. This puts high pressure on this document since a well thought through and described design hopefully will result in pieces matching together.

A solution is to create and debug all modules and submodules individually using test benches and waveforms to make sure the modules work exactly as they are supposed to before putting them all together.

3.1 The Logic of Adjusting Volume and Balance

This is a challenge. It will be solved.

3.2 Low Pass Filtering

Different approaches to making a function for a low pass filter will be considered.

The low pass filter will be a part of the `Analysis` module since it will only be used to refine the displayed power levels. The respective bars rendered on the screen will be delayed by at least 100 ms in order to allow a satisfactory result of the filtering, since there s a need for future values and the filtering is done in real time.

Before implementation, time will be put aside to study and understand the problem further.

3.3 Bar Graph Rendering

Stuff has to be rendered on the screen. This is not an issue, but rendering the right stuff is.

4. User Interface

The user interface will be able to display all the manageable settings on a VGA screen. There will in total be four bars. One to indicate the left incomming power, one to indicate the right incoming power, one to indicate the modified left power and one to indicate the modified right power.

The user interface will also display the current volume graduated in dB. The scale goes from -15 dB to +15 dB. This is controlled by the arrow keys, up and down. The balance indicator appears at the bottom of the interface. The balance indicator works like 0 is equal to the same amout of power from both left and right, if you press the right arrow at the keyboard, the balance indicator will step up the right side of the "0".

There will also be a mute figure to show if the mute button is activated.

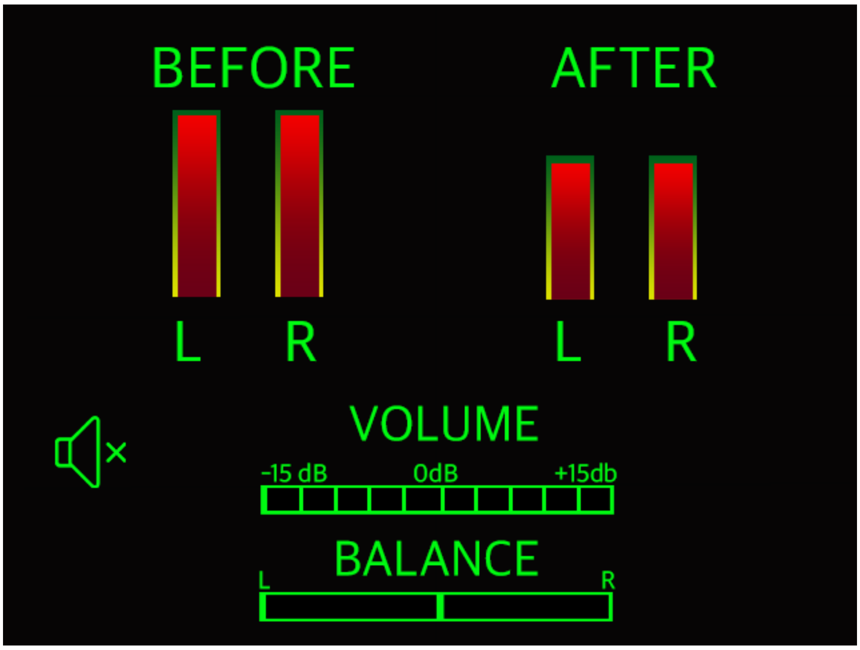

Figure 4.1: User interface