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

Group 41 - Design Specification

Högskoleingenjörsutbildning i datateknik, 180 hp

Design Specification - September 30, 2015
System Design - Project, HT15
TSIU03, Linköpings universitet

Supervisor:
Petter Källström
Department of Electrical Engineering (ISY)

Contents

1 Introduction **1**

2 System Level Description **2**
2.1 Keyboard . . . 2
2.2 Snd_Driver . . . 3
2.2.1 Snd_Driver:Channel_Mod . . . 4
2.2.2 Snd_Driver:Ctrl_Block . . . 4
2.3 Vol_Bal . . . 4
2.4 Analysis . . . 5
2.5 VGA_driver . . . 5
2.5.1 VGA_driver:Pipelining . . . 6
2.5.2 VGA_driver:pixelcounter . . . 6
2.5.3 VGA_driver:linecounter . . . 7
2.5.4 VGA_driver:clock_divider . . . 7
2.5.5 VGA_driver:blank_video . . . 7
2.5.6 VGA_driver:RAM_control . . . 7
2.5.7 VGA_driver:hsyncr, vsyncr . . . 7
2.5.8 blank_syncr . . . 7
2.5.9 VGA_driver:pixel_reg . . . 7
2.5.10 VGA_driver:vga_gen . . . 8
2.5.11 VGA_driver:bartender and barmixer . . . 8

3 Challenges in the Design and Proposed Approach **9**
3.1 The Logic of Adjusting Volume and Balance . . . 9
3.2 Low Pass Filtering . . . 9
3.3 Bar Graph Rendering . . . 9

4 User Interface **10**

A Appendix: System Overview **I**

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. A high resolution version is also included in *Appendix A: System Overview*.

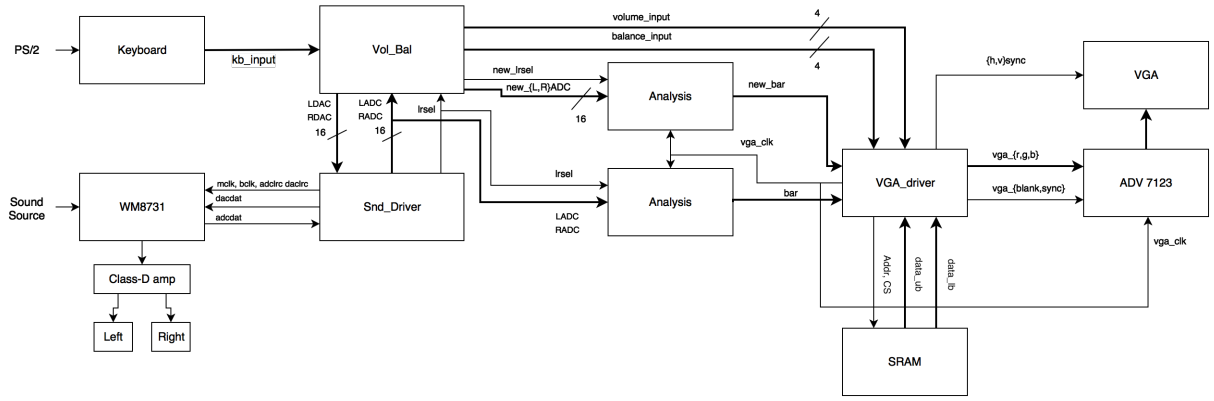

Figure 2.1: A graphical overview of the system's first module layer. Letters inside curly braces indicates multiple signals exlusively including each of the letters.

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 and corresponding `kb_input`

KEY	MAKE	BREAK	`kb_input`
U ARROW	E0,75	E0,F0,75	00001
L ARROW	E0,6B	E0,F0,6B	00010
D ARROW	E0,72	E0,F0,72	00100
R ARROW	E0,74	E0,F0,74	01000
END	E0,69	E0,F0,69	10000

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. `Snd_Driver` consists of two submodules: the `Ctrl_Block` and two instances of the submodule `Channel_Mod`. Depicted below (Figure 2.3), is a graphical representation of `Snd_Driver`.

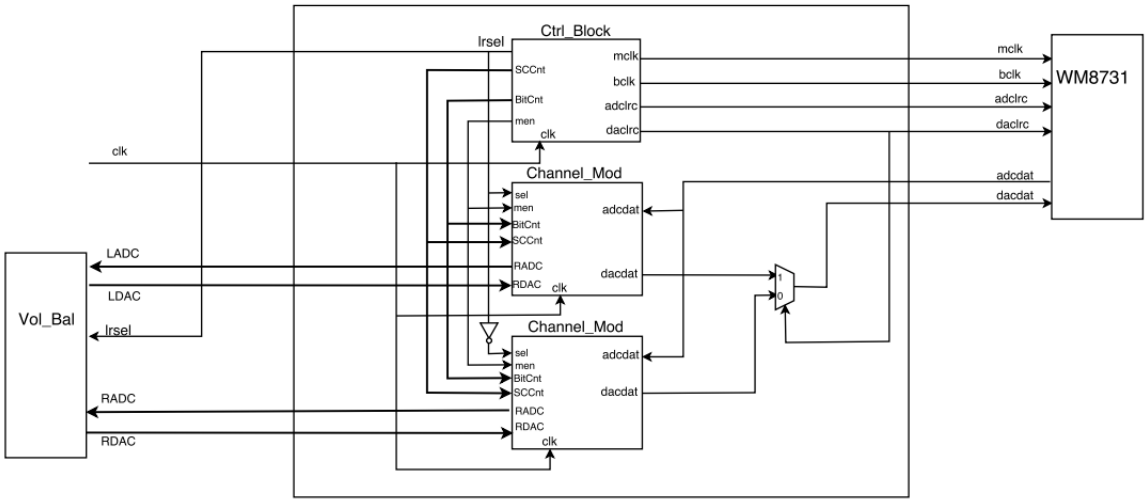

Figure 2.3: Graphical representation of Snd_Driver.

Figure 2.4: List of output signals

Name	Description
`LADC/RADC`	Left/right outgoing samples for the `Vol_Bal` module to process. 16-bits size.
`lrsel`	Select signal for usage of left/right sample.
`mclk/bclk`	`mclk` is WM8731's internal operation clock, bclk is a bit clock
`adclrc`	Left/right selector for `adcdat`. `adclrc`='1' for left.
`daclrc`	Left/right selector for `dacdat`. `daclrc`='1' for left.
`dacdat`	Serial bits to the DAC, one bit per `bclk` pulse.

Figure 2.5: List of input signals

Name	Description
LDAC/RDAC	Left/right incoming samples which shall be shifted out to the WM8731. 16-bits size.
adcdat	Serial bits from the ADC, one bit per bclk pulse.
clk	The system clock. (50 MHz)

2.2.1 Snd_Driver:Channel_Mod

Channel_Mod is a submodule instantiated twice, once for each bidirectional channel. One for the left and one for the right channel. The difference between the two is the sel, or select, signal. One instance receives the sel signal inverted. Channel_Mod gets the sel signal, which if active, indicates that the sound has been processed by Vol_Bal for the selected channel and is ready to be sent back to WM8731. Depending on Sel, Channel_Mod shifts in/out the bits from/to adcdat/dacdat.

2.2.2 Snd_Driver:Ctrl_Block

Ctrl_Block acts as the control block of the module. It is responsible for generating several control signals. The module is built upon a 10-bit counter, cntr. The control signals for the rest of the module is generated from different bits of cntr in the Ctrl_Block, essentially keeping track of what shall be done at which time.

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 SndDriver 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.

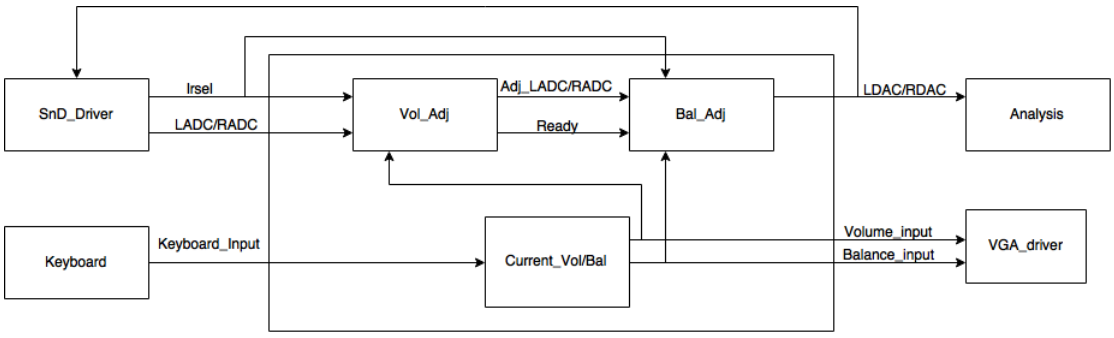

Figure 2.6: An overview of the Vol_Bal module's internal workings.

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 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 described in table 2.8, and five output signals described in table 2.9. It consists of eleven sub modules which can be over viewed in 2.7 described below.

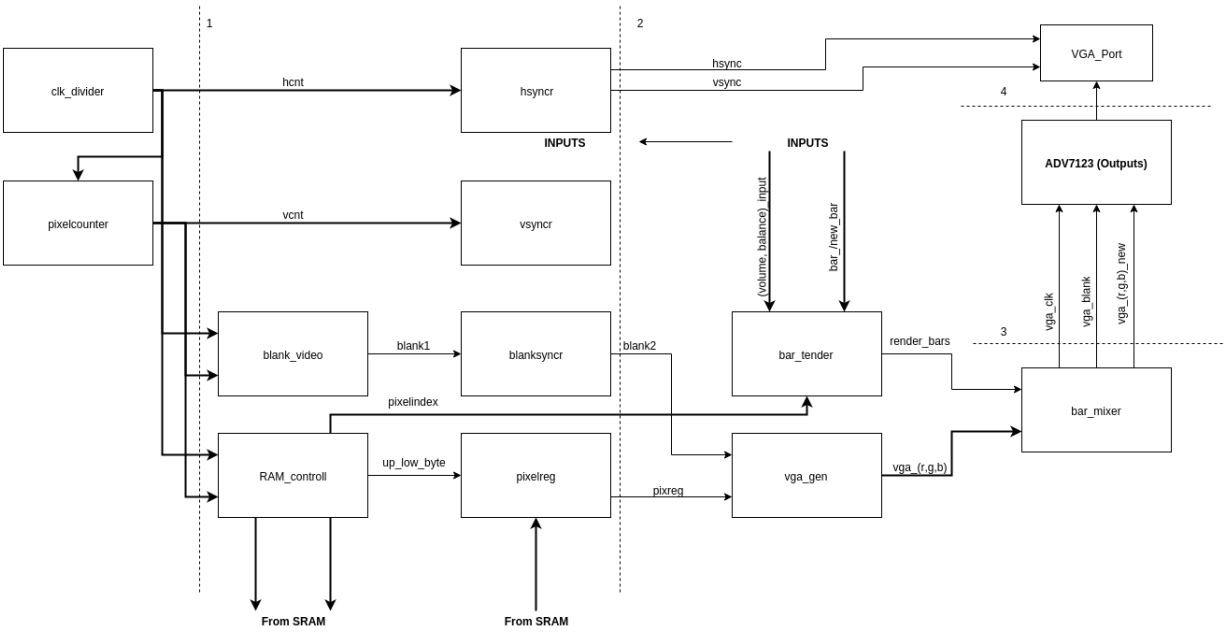

Figure 2.7: Block diagram of `VGA_driver`

Figure 2.8: List of input signals

Name	Description
`volume_input`	a 4 bit input containing volume information
`balance_input`	a 4 bit input containing balance information
`bar`	a n bit input containing signal sound input signal level
`new_bar`	a n bit input containing manipulated input signal level

Figure 2.9: List of output signals

Name	Description
`vga_clk`	clock signal needed for scanning
`vga_blank`	a blanking signal for blanking when resetting scan
`vga_(r,g,b)`	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 2.7. 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:pixelcounter

The sub module `pixelcounter` is a generating the internal 10 bit signal `hcount`. `hcount` functions as a `pixelcounter` for each row and will count from 0 to 797 which represent the pixels needed for each row in the scanning.

2.5.3 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.

2.5.4 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

2.5.5 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.

2.5.6 VGA_driver:RAM_control

The image used as a background is pre-stored in the SRAM and to acess the image data we need to generate some signals to the SRAM. The control signals `CE, OE, WE, UB, LB` are set constant to 0, 0, 1, 0, 0, which means that SRAM should be enabled read-only and access both upper and lower byte.

Each 16 bit row in the SRAM contains color information about two pixels, one in the lower and one in the upper byte. Therefore we must generate two signals: `sram_addr` which provides the SRAM with the correct adress and `up_lo_byte` that provides the sub module pixelreg with the information about which of the bytes should be read . It also generates the output `pixelindex` which is a counter that counts each pixel in the visible area used by `bar_tender` (see section 2.5.11).

2.5.7 VGA_driver:hsyncr, vsyncr

The sub modules `hsyncr` and `vsyncr` are responsible for generating the `hsync` and `vsync` signals used for resetting the sweep. In the resolution used in this application this means that `hsync` should be active(low) during hcount values between 490 and 493 and `vsync` between `vcount` values of 655 and 750.

2.5.8 blank_syncr

`blank_syncr` is a single d-flip flop needed for pipelining.

2.5.9 VGA_driver:pixel_reg

`pixelregister` is responsible for reading the image information from SRAM, the image information is avaliable on the SRAM-bus and the module simply needs to read the correct byte (using `up_lo_byte`) and put it in a pipeline register.

2.5.10 VGA_driver:vga_gen

vga_gen will take the pix_reg register and supply it to the three color channel outputs vga_r, vga_g and vga_b. Due to the fact that the AVD7123 chip expects 10 bit values and the stored image containing 3-bit color values the values need to be scaled up before provided to bar_mixer

2.5.11 VGA_driver:bartender and barmixer

bar_tender 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, bar_tender will calculate which pixels should be blanked and set the signal render_bar high.

bar_mixer 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.

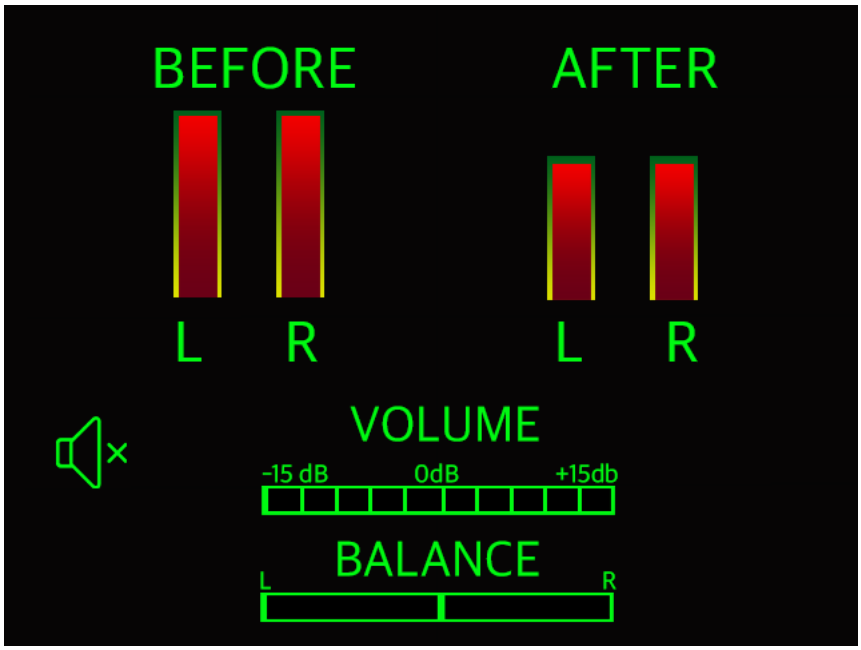

Figure 4.1: User interface

A. Appendix: System Overview