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

Group 41 - Project Report

Högskoleingenjörsutbildning i datateknik, 180 hp

Project Report - November 4, 2015
System Design - Project, HT15
TSIU03, Linköpings universitet

Supervisor:
Petter Källström
Department of Electrical Engineering (ISY)

Contents

1 Introduction

This is the final report treating the final project in the course TSIU03 System design. The project was to design and implement a device used for audio signal processing.

Using a DE2 board an external audio source can be connected via a 3.5 mm port, and accepts input for volume, balance and mute settings via a PS/2 keyboard the signal is then processed by the application and an image is rendered on a VGA-monitor displaying four signal level bars representing the left and right channels both before and after manipulation. There is also one bar displaying the current volume and one bar displaying balance setting. There is also a symbol showing if the system is muted.

This report covers all aspects of the work involved in the project. It contains a brief system description, it lists the major challenges and solutions to said challenges and personal experiences from the group members involved.

2 Achievements

2.1 User Noticable Achievements

- Remarkably astonishing graphics.
- Power level indicators for both the input and output sound signal.
- The indicators are moving at a smooth rate with absolutely no flickering.
- Displaying the current volume and balance adjustments made in a user friendly way.
- Both the minimum and maximum level of adjustments available to the volume and balance is visible to the user.
- A mute button is shown if the volume is set to be muted.
- Output sound is sent to a class-D amplifier.
- Each bar has a peak level indicator.
- The peak level indicator falls slowly when the power bar is lower than the peak level indicator.

2.2 Design Challenges

- Use the SRAM to store information about a background picture.
- Implement a low-pass filter for our signal level indicators.
- Bar graph rendering (which pixels to write/blank).
- The logic in adjusting the volume and balance correctly.
- Detect each key press only once to prevent drastic changes.

3 System Level Description

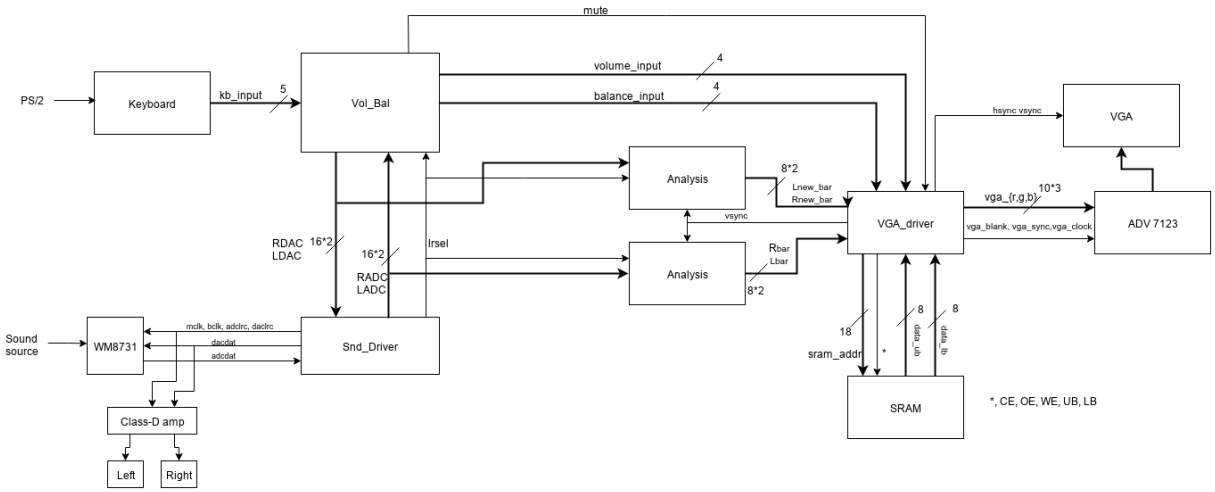

Figure 1: G41 Project System Overview

The project consists of five major modules – `Keyboard, Vol_Bal, Snd_Driver, Analysis,` and `VGA_Driver`. Presented here is an overview of the modules, and detailed functionality is further described in the *Design Specification [1].*

The system takes, with help of `Snd_Driver`, digitally encoded sound from the WM8731 chip. The sound data is passed on to the first instance of `Analysis` which provides (in order to smoothen the bar movements) low-pass filtered control signals to `VGA_Driver:bartender` to assist the rendering of the pre-processing bar graphs done by `VGA_Driver` as a whole. The audio data from `Snd_Driver` is also passed on to `Vol_Bal` which adjusts the audio balance and volume according to register values set by the control signals from `Keyboard`. The adjusted audio signals are then sent back to `Snd_Driver` which sends signals for the class-D amplifier through the GPIO:s. The adjusted audio is also forked off to a second `Analysis` instance, which assists the post-processing bar graph rendering.

Whereas `Keyboard` and `Analysis` already are fairly small and monolithic modules, and `Snd_Driver` and `Vol_Bal` consists of three submodules each, `VGA_Driver` consists of no less than twelve submodules.

`Keyboard` handles the PS/2 keyboard user input. The module filters break scan codes ($F0_{16}$, XX_{16}) bytewise (11 bit/byte) and compares the XX_{16} byte (*iff* directly preceeded by $F0_{16}$) to the control key values. This approach will allow the system to respond to the regular keys (ARROW keys and END) as well as corresponding numeric keypad keys as initial $E0_{16}$ bytes are discarded. Upon a registered valid key release, a 5-bit control signal (`kb_input`) is sent for a single clock cycle to `Vol_Bal`.

`Analysis` reads the `ADC` signals and lowpass filters these with a saturation time of approximately 100 ms (4096 clock cycles). These signals are converted to a graph height (in pixels) and passed on to `bar_tender`.

`Snd_Driver` have the three submodules `Snd_Driver:{Ctrl,Channel_Mod}` if we count the two instances of `Channel_Mod`. These are verbatim copies of *Laboration 4*. The only difference is

that the outputs are sent to GPIO pins (and from there via an i2c adapter to the class-D amp) as well as back to the WM8731 codec.

`Vol_Bal` consists of the three submodules `Current_Vol_Bal, {Volume,Balance}_Adjustment`. `Current_Vol_Bal` reads `kb_input` and updates registers representing system levels of volume, balance and mute. `Volume_Adjustment` adjusts volume according to the `volume_level` input from `Current_Vol_Bal`, decreasing amplitude by up to -30 dB in 3 dB decrements, and `Balance_Adjustment` adjusts the audio according to system balance level, resulting in a linear scaling of the incoming amplitude. The stronger a channel bias, the higher the signal reduction of the opposing channel. The affected channel loses 1/8 of the amplitude per level of bias.

`VGA_Driver` with its submodules handles the rendering of the UI. The module is, in essence, a modified version of the module used in *Laboration 3*. The most significant difference is found in the two new modules `VGA_Driver:Bar_Tender` and `VGA_Driver:Bar_Mixer`. `Bar_Tender` uses the output from `Vol_Bal` and each of the `Analysis` modules along with `{h,v}cnt` to calculate which pixel is to be rendered and if it should be rendered from the background image or if to draw it black. The result of the calculations is the control signal `render_bar`. This signal is then passed to `Bar_Mixer`, which is basically a multiplexer, either forwarding the loaded background colours or a blacked out pixel depending on `render_bar` from `Bar_Tender`.

4 Justification of the Achievements

4.1 Justification Keyboard

In order to detect and register a keypress only once, the `Keyboard` module had to differ significantly from *Laboration 2*. Two viable options were considered: Either, a control for typematic rate effectively reducing the polling rate, else a "detect-on-release-only" would have to be implemented. The latter approach was decided upon.

The basic functionality of the module is the same as in the laboration, with the scan codes going through a shift register and being checked against a set of 8-bit values. These values correspond to the control keys plus the break byte $F0_{16}$.

When a start bit reaches the end of the shift register, the shift register is set to reset to 1:s next `PS2_CLK` cycle, allowing us to detect each new byte as they are shifted through. This results in the shift register being "unusable" for one `PS2_CLK` cycle. We circumvent this problem by using a 10 bit (1 bit less than a full sent byte) and ignoring the stop bit.

Next, the current 8-bit value following the start bit is checked. If it equals $F0_{16}$, a register `BREAKSET` is set in order to enable the next byte to be checked against the control key values. If at the next byte cycle `BREAKSET` is not set, the byte is ignored. If set, `BREAKSET` is reset to '0', the byte checked and `kb_input` set to corresponding value.

"Outside" this `PS2_CLK` cycle, the process is controlled by the regular clock. This allows us to again set `kb_input` to idle (all '0') each system clock cycle, which in turn allows us to restrain `kb_input` to be anything but 0 for no more than 1 clock cycle and therefore never registered by `Vol_Bal` more than once per key press.

4.2 Justification VGA

One challenge was the image rendering on the VGA-screen. Since a pre-stored background image is being used, it was decided that the easiest way to implement the signal level indicators with a gradient was to create a background image containing filled bars and render black over the parts that should not be filled rather than having the application draw the bar itself.

Figure 2: Concept UI image

Figure 2 is a concept version of the image used. Constants were declared using the coordinates of the upper right corner of the bars displaying signal level, both right and left channel before manipulation, the volume bar and the mute symbol and the width height and offset to the remaining bars. This way conditions could be defined for when `hcnt` and `vcnt` coordinates were inside the bars or not.

The input consists of four 8 bit unsigned with values spanning from 0 to 171 that holds information about the signal level bars, a 4 bit unsigned for volume setting spanning from 0 to 9 and a 5 bit signed value for balance setting spanning from -8 to 8.

For the signal level bars the application checks that `hcnt` and `vcnt` is in the bar and then compares the `vcount` value with (171 – input) (which will represent the part of the bar that should be painted over). And if `hcnt` and `vcnt` is in the correct area the output `render_bar` is set high and black will be rendered in the affected pixel.

The volume bar works in a similar way but the input is multiplied with a constant declaring the width of the boxes, and checking towards `hcnt` instead since the bar is positioned horizontally rather than verticaly.

The balance works as volume with the exception that it is divided in two parts, one representing the right side being filled and one representing the left.

There is also support for peak level indicators. They are represented by 8 bit unsigned internal signals that are assigned the bar value iff the current bar value is greater than the last value

assigned. A counter counting 50k clock cycles is then decremented before the peak level amplitude is decremented. The application draws the peak level line in the same manner as for the bars, but setting the output `render_peak` high instead to give the peak level indicator a different color than the one generated from `render_bar`.

Since the application used in *laboration 2* also rendered a background image stored in the SRAM only minor ajustments needed to be done. The sub module `bar_tender` described above sets `render_bar` and `render_peak` high in the affected pixels and the submodule `bar_mixer` works as a multiplexer forwarding color-information from SRAM when `render_bar` and `render_peak` are set low. And forwards black (for bars) and white (for peak) when the inputs are set high.

The mute symbol is implemented simply by drawing a black square over the symbol whenever the `mute` input is set low.

4.3 Justification Analysis

One challenge that we had through the project were to implement a low-pass filter to get smooth moving bars. During the design part we spent a lot of time thinking of how to do this and calculating math.

The incoming signals are low-pass filtered with a saturation time of approximately 100 ms (4096 sample cycles) as seen in figure 3, resulting in a measurement of the signal's power.

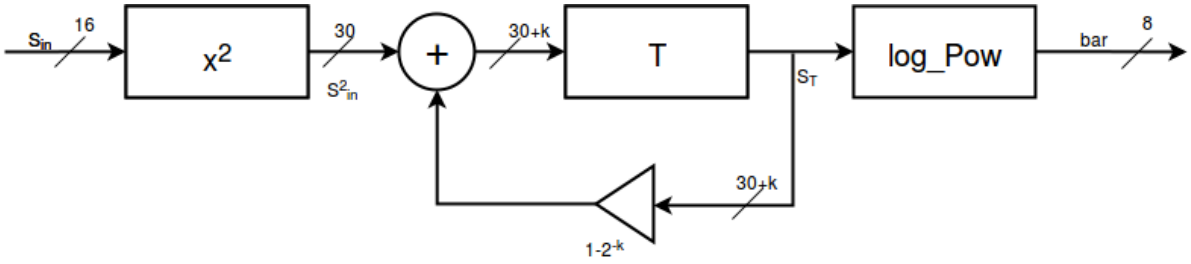

Figure 3: The low-pass filter. k is chosen by the approximation $\frac{1}{10}\ \mathrm{s} = 2^k \cdot \frac{1}{48800} \Rightarrow 2^k = 4880 \approx 2^{12} \Rightarrow k = 12$

`log_Pow` takes the logarithm of the low-pass filtered signal. It provides module outputs proportional to the result, updating only in sync with `vsync`. The `bar` signals are lined to `VGA_Driver` as definitive information about how the power bars should render on screen.

4.4 Justification Volume and Balance Adjustment

The framework for the layout of the `Vol_Bal` module and its functionalities is grounded in item 5 and 6 of the *Requirement Specification [2]*. Design choices regarding volume and balance adjustment were made with those in mind, and one of the first steps was to centralize these functionalities within one module, thus, `Vol_Bal`. Inspiration was drawn from the system structure in *Laboration 4, Audio Codec* where this module replaces the `Application` module.

Two serial and separate submodules were decided on to handle audio signal adjustment - first a logarithmic adjustment for volume, which is then followed by linear balance scaling. The balance adjustment was initially interpreted as being logarithmic, but corrections were made which led to the splitting of the two. A third submodule in `Vol_Bal` was assigned to read the

`kb_input` signal and determine when user input is legal and then updating internal system levels accordingly. It was also decided that mute would be treated as a separate system status instead of implementing a special case for "no" volume. Since volume adjustment represents logarithmic reductions in the amplitude, we would need a workaround to truly mute output anyway.

A natural increment/decrement in decibel to scale the volume is ± 3 dB, since a 3 dB increase of a signal is approximately twice the power or the amplitude multiplied by the square root of 2. Furthermore, since the incoming signals can be assumed to use the full range of the amplitude sampling, it makes sense to only make signal adjustments that lower the amplitude output. These facts led us to the decision to allow for eleven levels of volume, where internally an unsigned value between 0 and 10 represents the amount of logarithmic reductions we perform on the incoming amplitude, per the formula below. The result is a possible volume adjustment on the range -30 to 0 dB, with intervals of 3 dB.

$$A_{adj} = A_{in} \cdot (1/\sqrt{2})^n$$

An early design choice was to avoid division in general unless performed by a `shift_right` statement (allowing for divisions of powers of 2). This affected the algorithm for the linear balance scaling. Since the requirements document specified at least ten levels of balance, the conclusion was to subtract the incoming signal with one-eights to implement the formulas below:

$$A_{l\_out} = \frac{8-m}{8} \cdot A_{l\_adj} \quad , \; A_{l\_out} = A_{l\_adj} \; for \; m < 0$$

$$A_{r\_out} = \frac{8-|m|}{8} \cdot A_{r\_adj} \quad , \; A_{r\_out} = A_{r\_adj} \; for \; m > 0$$

After a few iterations, the above was locked in with 17 legal values for m, allowing for complete muting of a separate channel.

4.4.1 Implementation

To realize the logarithmic volume scaling, a varied amount of multiplications needs to be performed on the incoming signal. To achieve this, a state machine for its submodule was adopted, utilizing the steps below:

- `idle:` Set `volume_done` to low. When the `lrsel` signal changes, load the relevant audio signal and the volume system level, and only then go to state `odd`.
- `odd:` If the volume input is odd, multiply the signal by 3 and divide by 4 (0.75 = approximation for $1/\sqrt{2}$). Go to state `evens` in either case.
- `evens:` If the copied volume input is 2 or more, half the amplitude, subtract 2 from the volume input, and go to state `evens`. If not, go to state `end`.
- `end:` Set `volume_done` to high. Demultiplex the result to the left or right channel, using `lrsel` as a selection signal. Go to state `idle`.

We only want to adjust for balance once we are done with the volume part. This led to the logical inclusion of the `volume_done` output from the serially preceeding module. In line with the mathematical functions above, we select our balance value based on the `lrsel` logic and

whether the balance level is positive or negative. Multiply by 8 minus the balance value, followed by a shift of three bits right to divide by 8. If `lrsel` or balance level indicates we are not to balance adjust the current audio channel, balance value will be 0, resulting in status quo since we multiply by 8, and then divide by 8 (the order is important).

From the design, `Current_Vol_Bal` module's task is clear. Read the 5-bit wide `kb_input` for issued commands and update system level registers accordingly. The implementation first uses simple `AND/OR` logic to check whether the command is legal based on our current system levels, zeroing out any illegal `kb_input`. The registers update every clock cycle, incrementing or decrementing whenever the corresponding bits in the modified `kb_input` signal is a '1'. Mute is `XOR`'d with its current value and mute-bit in `kb_input`.

5 User Interface

The system is controlled by five keys on a PS/2-connected keyboard. Down in the table you can see which key that changes what in the system.

KEY	Function
U ARROW	Volume Increase
L ARROW	Balance Bias Left
D ARROW	Volume Decrease
R ARROW	Balance Bias Right
END	Mute Volume

Figure 4: PS/2 keys and how the input affects the system.

The volume level has eleven different stages, exclusive mute. When the volume is at max, the whole bar will be red, and when the volume is decreased by one the bar will decrease to show the current volume.

The balance level has eighteen different stages. Eight for left balance, eight for right balance and one stage when the signal is equal. When the balance is equal for left and right the only thing you will notice is the bar and divider. But when you for example press the left arrow, the left side will be filled and the outgoing volume on the right will decrease.

To indicate that mute is enabled you can see a green speaker with a cross to the left side of the picture, and when mute is disabled there will only be a black fiel. Apparent is that the "after" bars will not show anything, due to nothing is being sent out.

You will be able to see one peak level indicator for each bar, the peak level indicator falls slowly when the power bar is lower than the peak level indicator. With our peak level indicator our user experience is taken to a whole new level, as you will notice when you use our system you will be overwhelmed by the effect.

6 Improvements

The system follows the requirement and design specification quite closely. However there are still possibilities to further develop it into an enjoyable piece. The peak level indicator, feature

Figure 5: User interface

wise, could when updated have a slight pause at the peak, so it can be observed more clearly before beginning to drop.

The bars of the sound amplitude are currently a gradient drawn on the background image, being covered by a black bar, to display the amplitude in a gradient bar. However, since it is drawn in the image of the background, the gradient can not change. A possible improvement to the aesthetic of the bar would be rendering the gradient with every time the bar is rendered. Of course this requires quite a bit of back tracking because it is a vastly different way to draw bars.

Improvements to the volume and balance bars could include adding more levels. It would be an easy improvement to make, seeing as it is a change that does not require going back on decisions already made. The balance bar would of course require a slight change in the background image because of the pre-drawn boxes for the balance levels. The volume bar does not have the same limitation.

From the keyboard, we currently read the button being released. This helps us avoid problems with the same button being pressed multiple times, or held down. Therefore, you have to press the button multiple times to cause the effect of it to happen multiple times. If you for example would like to switch the balance fully to the left channel from being fully to the right, that is ten key presses. With clever PS/2 handling, support could be added for a button being held down in order to repeat the command of that button. This would especially be useful if more levels were added to the volume and balance levels.

As an improvement, the keyboard could be given control of a lot of variables in the system. For example, the coefficient of the lowering of the peak level indicator, the low-pass filtering coefficient (in `Analysis`). This would give the user more control of the system.

7 Evaluation of the Project Execution

Time spent for each member, and respective member's tasks:

Group Member	Main Tasks	Total Time
Niklas Blomqvist	Analysis code	∼65
Philip Johansson	Analysis code	∼70
Matteus Laurent	Vol/Bal code, project coordinating	∼75
Johan Levinsson	Top TB code	∼55
Oscar Petersson	Keyboard code, documentation	∼70
Erik Peyronsson	VGA code modifications	∼70
		405 h

The overarching theme of the project execution would have to be the difficulties faced when one or more members' participation was prevented by illness or similar obstacles, ensuring a less than ideal attendance. Our project group was set back early on because of this very reason, leading to us trailing the schedule by approximately a week. However, the effort invested to catch up was admirable and sufficient, allowing for the eventual completion of the project. Aside from this, the project went mostly according to plan, with a few notable exceptions.

Cooperation and coordination within the group was satisfactory. The project manager was given sufficient authority to lead the distribution and coordination of tasks. A certain level of trust combined with good design of the module overview allowed for more individual workloads, without sacrificing shared communication and system understanding.

All things considered, the design specification was the task that exceeded its set aside time the most. This was due to the document outlining a lot more module details than originally planned, ultimately giving an indepth view of the system. The document was not only expanded upon on recommendation from our project supervisor, but also because of our own natural inquiry when discussing the design overview. Laying the groundwork this way gave voice to design uncertainties early on, and thus helped a lot with subsequent tasks.

Thanks to a thorough design specificiation, the foundation for our first prototype essentially wrote itself. The initial stages of coding were completed swiftly, making up for any time lost (and more) on writing the design document.

8 Personal Experiences

8.1 Niklas

With a well written design the conversion from text to VHDL went super smooth and was therefore very enjoyable. One thing I will take as a lesson is the fact that you should consider which different ways of implementing the same problem there is, consider the pros and cons of each way and from there on pick the most suitable one. I definitely had a good time working on this project.

8.2 Philip

We spent a lot of time on design. We discussed a lot in the group and calculated on much, the whole design was very well thought out and I an had very good control throughout the system. This led to the implementation going very well, and there not being much trouble in assembling the system. This is something I will take with me in life.

8.3 Matteus

All things considered, the project was enlightening with many valuable lessons. Translating the design into our first prototype was quick and done with ease thanks to our design specification - thus emphasising the importance of preparation and good structuring.

Some specific knowledge learned: Modelsim is a powerful tool in resolving bugs and ensuring correct design implementation; word lengths and overflow are the sources of many problems; timing diagrams are very useful.

As for improving the course - we were advised to not use a certain state machine implementation shown in the lectures.

8.4 Johan

The most enjoyable part for me was seeing the shift from design specification to VHDL code. In my head it would not go as easily as it turned out to. If there is anything I wil make sure to learn from, it is going to be writing good design speicifcations. I feel that it should be applicable to other projects.

8.5 Oscar

Foremost, the project has significantly increased my comprehension of digital circuit design. It has also shown that (which I am sure all of us already knew) that preparation is indeed key to success. Considering my fetish of documentation, I can not truthfully say that the sparse form of documentation has been a favourite of mine, but considering the given time frame, the intention with the project, and the fact that the course runs in parallel with TDDI02, I have to give it a pass. Over all, it has been a greatly enjoyable course.

8.6 Erik

As previous members have noted the design was very well thought through and the implementation only differed in minor areas. My biggest challenge was to operate the software, modelsim especially. I probably should have payed more attention in the labs on how to operate it cause getting it started and getting the test-bench working took almost as long as the testing itself wich felt like a waste of time.

9 References to the Project Files

[1] Group 41 - Design Specification, *Linköping*, 2015-10-19.

[2] Group 41 - Requirement Specification, *Linköping*, 2015-09-22.

[3] Group 41 - First Presentation, *Linköping*, 2015-10-15.

[4] Group 41 - Project Plan, *Linköping* 2015-09-24.

All documents available at https://github.com/oscpe262/TSIU03.Project. The references in the PDF version of this document are hyperlinks to the document listed. The PDF can be downloaded at: https://github.com/oscpe262/TSIU03.Project/blob/master/Projektrapport/report.pdf.