Project Report

TSIU03

Students:

Nell Party, Apollinaire Criquet, Omid Najafi, Johan Klasén and Henrik Lindgren

supervisor:

Petter Källström

Table of Contents

Introduction

This report describes a system modulating and analyzing an audio signal and producing output on a VGA-monitor. The system can control the volume and balance of its input signal based on input from a PS2-keyboard, outputting the resulting audio while at the same time analyzing the frequency components of the signal. The VGA-monitor displays the power of each of the analyzed frequency bands as well as the currently set volume and balance. An oscilloscope type plot of the input signal is also shown on VGA-screen.

Design

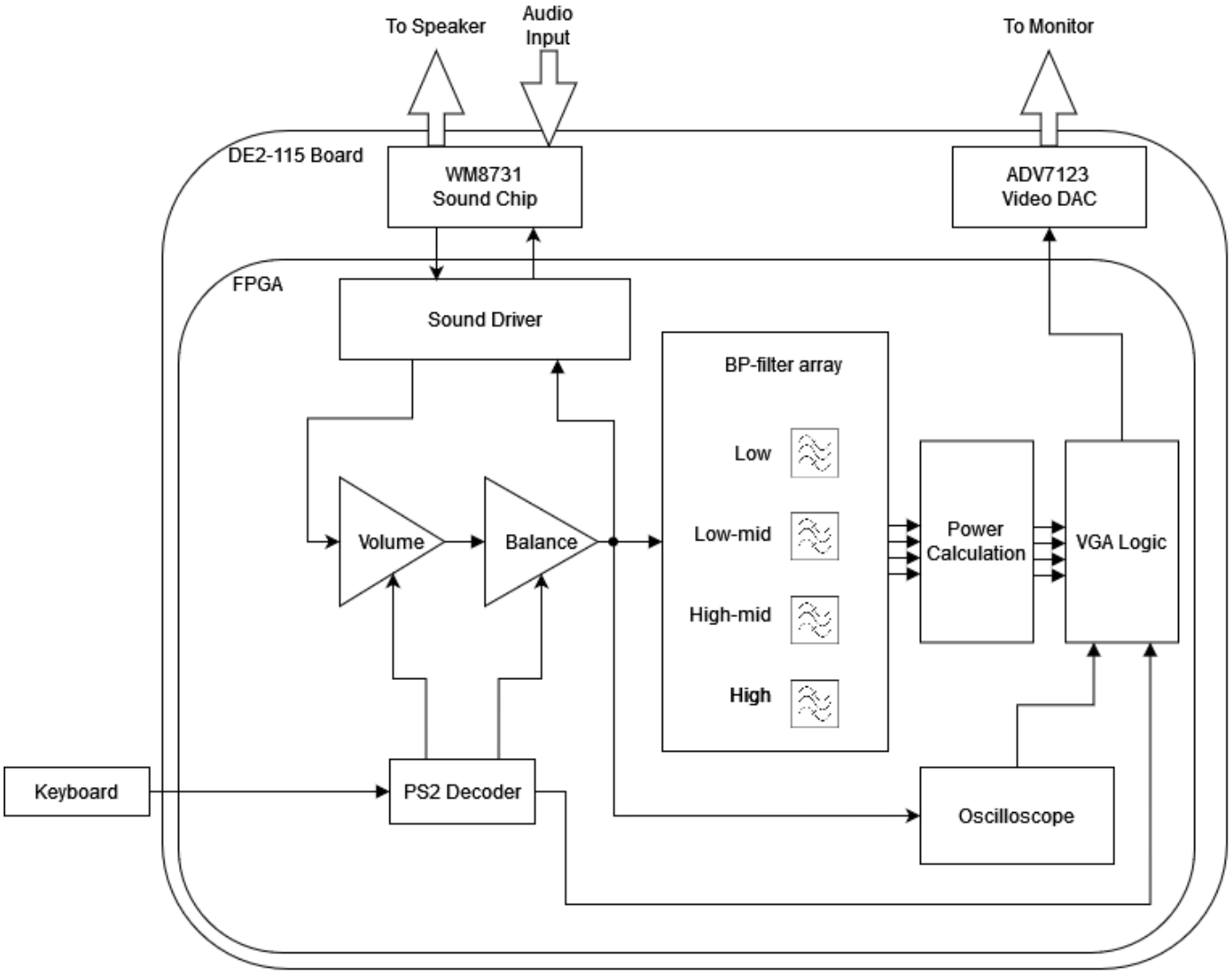

Figure 1: Overall system block diagram

The block diagram in figure 1 shows the overall design of the product. To make it easier to follow the design, the project is divided into seven separate modules called *PS2 Decoder, Sound Driver, Volume and Balance Control, Filter Bank, Power Calculator, Oscilloscope* and *VGA Logic*.

In a very short overview of the system the main signals of the device are the digital audio samples which are sent from the *WM8731 Sound Chip* to the sound driver. The Sound Driver is responsible for timing the samples correctly according to the sample rate as well as alternating between the left and right sound channels. The audio samples are passed along into the Volume and Balance Control and gets modulated according to respective volume and balance levels which can be changed with keyboard presses with the help of the PS2 Decoder. The volume and balance adjusted samples are then sent back to the sound driver which passes it on to the sound chip where the samples are converted back to analog signals which can be output to a speaker.

However, the adjusted digital samples are also passed along to the Filter Bank and the Oscilloscope. The Filter Bank splits the audio samples up into four separate frequency bands and into the Power

Calculator which determines the total power for both the left and right channel of all the four frequency bands. These values are then converted into a logarithmic scale which can be drawn and displayed on a screen by the VGA Logic module. The Oscilloscope also takes in the combined adjusted samples and calculates values which can be used by the VGA Logic to draw a waveform of the sound on the background of the screen.

Communication

The communication between the modules is done with the different signals which they take in input and the one they give in output. As it is shown in the block diagram:

- The Keyboard sends to the PS2 decoder two signals of one bit, one is the data line which serially sends one bit of information on the last key press, and the other to give the clock of the keyboard. After the PS2 decoder will interpret the signals to know which key had been pressed.
- The PS2 decoder stores the volume and balance settings and changes them based on the key-strokes received. The output of the PS2 decoder has two signals of four bits, which will go to the volume and the balance control. A one-bit signal goes to the sound driver which sets if the sound should be muted. The signals also go to the VGA Logic so that they can be displayed on the screen.
- The sound driver gives and receives two signals of 16 bits, one for the left sample and the other for the right. These signals will be given to the volume and the balance control. They will modify the sample, and give it back to the sound driver, after they go to the speaker.
- The left and right samples from the volume and balance control goes into the filters and the oscilloscope part.
- The oscilloscope part and the VGA Logic will communicate, with two signals of ten bits. The VGA part with a *vcnt* signal, representing the line which is printed on the screen. The oscilloscope will answer by a *hcount* signal, representing where to put the dot of the oscilloscope for each line.
- In total, the filter outputs eight signals of 16 bit, four for the left part and the rest for the right one (because it has four different bands of filters). These signals will go to the power part to be treated.
- The power part will give sixteen signals to the VGA Logic, to print the power in output of each filter. It has one signal of five bits to do the logarithmic scale, and one of three bits to improve the image on the screen, for each output of the filter.
- The VGA Logic sends after all that the signals to the screen to print the image of volume, the balance, the power after each filter and the oscilloscope. It also sends five control signals and the adress to read in the SRAM memory.

Module description

In this chapter, each module is described in more detail.

PS2 Decoder

The PS2 Decoder is an extension of the module created in Lab2. It takes the serial data sent on the *PS2_DAT* and shifts it into a shift register according to the keyboard's internal clock *PS2_CLK*. Some additional logic matches the timing of the serial data according to a stop signal to make sure only a complete scan code byte is output to the 10 bits signal *scancode*. Keypresses on PS2 keyboards can however correspond to more than a single byte value, usually to distinguish between make and break codes. This is most commonly done by sending the byte values *F0* or *E0* before the byte corresponding to the actual key. For this reason, the two bits of highest significance in *scancode* are used as flags to indicate if the values *F0* and *E0* were received before the last byte value.

For this module's functionality five scan codes are used and translated into actions which can be seen in the table below.

Key	***Scan code***	***Action***
Up Arrow	E075	Increase the volume
Down Arrow	E072	Decrease the volume
Left Arrow	E06B	Shift the balance to the left speaker
Right Arrow	E074	Shift the balance to the right speaker
M	3A	Toggle mute on the sound output

A separate process is implemented to translate the scan codes and perform the appropriate action. This is done only once when a new scan code is received by the control signal *scancode_en* and for the arrow keys this is translated into an increment or decrement of the values stored for the current volume and balance setting as long as it does not put the value outside of its range. For the volume setting this value can range between 0 and 10, and for the balance setting this value goes from 0 to 8. The setting levels are then passed on to the other modules on the two 4-bit signals *vol_level* and *bal_level*. The *M*-key is just responsible for toggling a 1-bit control signal *sound_output_en* which is sent to the Sound Driver and decides if the audio samples are sent back to the sound chip or not.

Sound Driver

The Sound Driver is responsible for communicating with the DAC/ADC chip and to time the samples correctly according to the sample rate. It is a copy of the design from Lab4 with the addition of the one-bit control signal *sound_output_en*, mentioned above, for muting the audio output.

Volume and Balance Control

The volume and balance of the sound is adjusted by multiplying the incoming sample values by factors defined in tables in which the factor corresponding to a specific volume/balance is looked up. For the volume control both channels (left and right) will be multiplied by the same value, whereas for the balance control, the channels will be multiplied by different factors depending on if the balance is turned right or left.

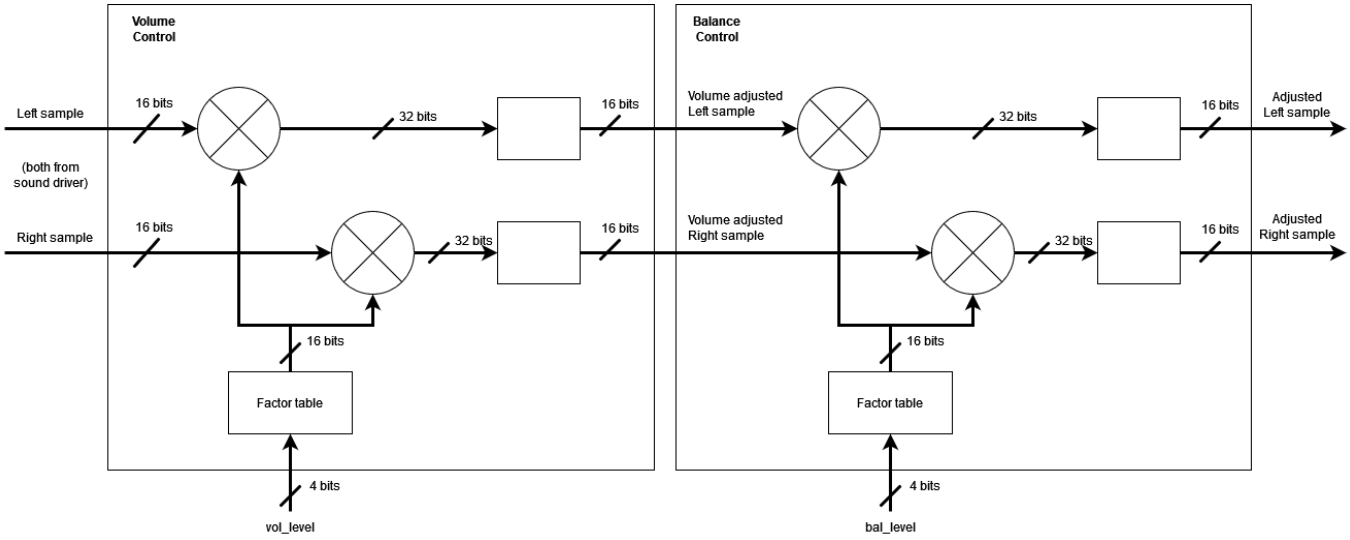

Figure2: Volume and balance control

Since the sample values should always be within the range of a signed 16-bit integer and to avoid using floating point arithmetic, some tricks need to be done when doing the multiplication. The factors will be values between 0 and 1 (from mute to full volume) so in order to get an integer factor we first multiply the factor by a power of two, discard the part after the decimal point and then divide the result by the same power of two. Using a power of two makes it efficient since we can use bit shifting for the calculation. The multiplications are pre-calculated and put into the tables of factors whereas the divisions consist of rightward shifting.

A logarithmic scale is used for the factors with 4 dB between each step and the values can be seen in the table below.

Vol level	**dB change**	**Factor**	**Hex value**
10	0	1.000000	8000
9	-4	0.630957	50C3
8	-8	0.398107	32F5
7	-12	0.251189	2026
6	-16	0.158489	1449
5	-20	0.100000	0CCC
4	-24	0.063096	0813
3	-28	0.039811	0518
2	-32	0.025119	0337
1	-36	0.015849	0207
0	-inf	0.000000	0000

Filter bank

The incoming samples (x_n) are stored in the internal RAM of the FPGA. Each filter bank (one for left channel and one for right channel) consists of 4 FIR-filters with different sets of coefficients which separate the signal into 4 different frequency bands. The FIR filters are designed in MATLABs Filter Designer which provides the coefficients needed for filtering the signals. The coefficients are also stored but in the internal ROM of the FPGA.

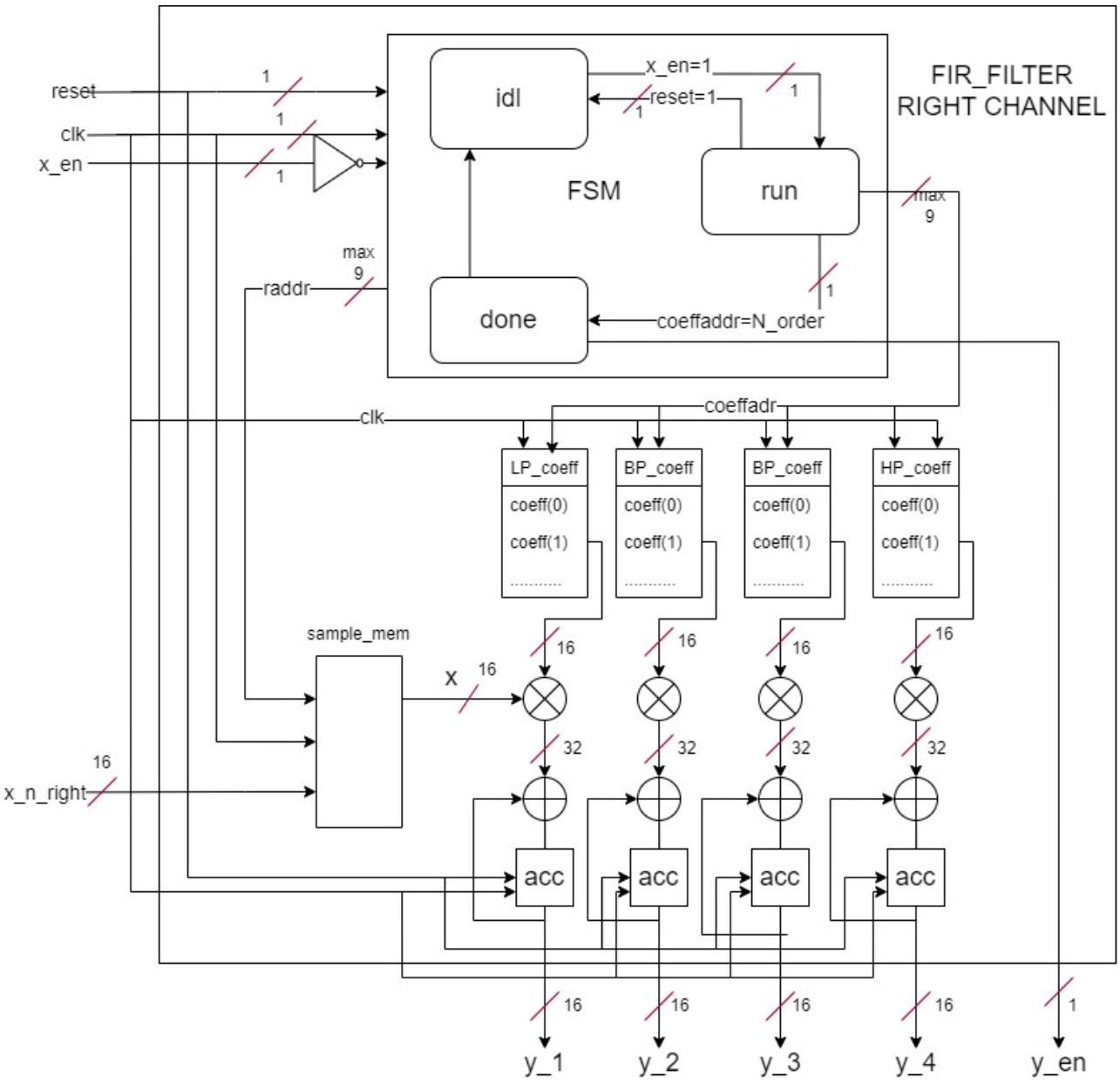

Figure 3: filter block diagram for the right channel. The numbers beside the red lines indicate the bits used for the signals

A FIR filter is defined by:

$$\boldsymbol{y[n]}=\sum_{k=0}^{N-1} h_k x[n-k]$$

• x[n] is the input signal.

• y[n] is the output signal.

• h_k is the k^{th} filter coefficient.

• N is the length of the filter.

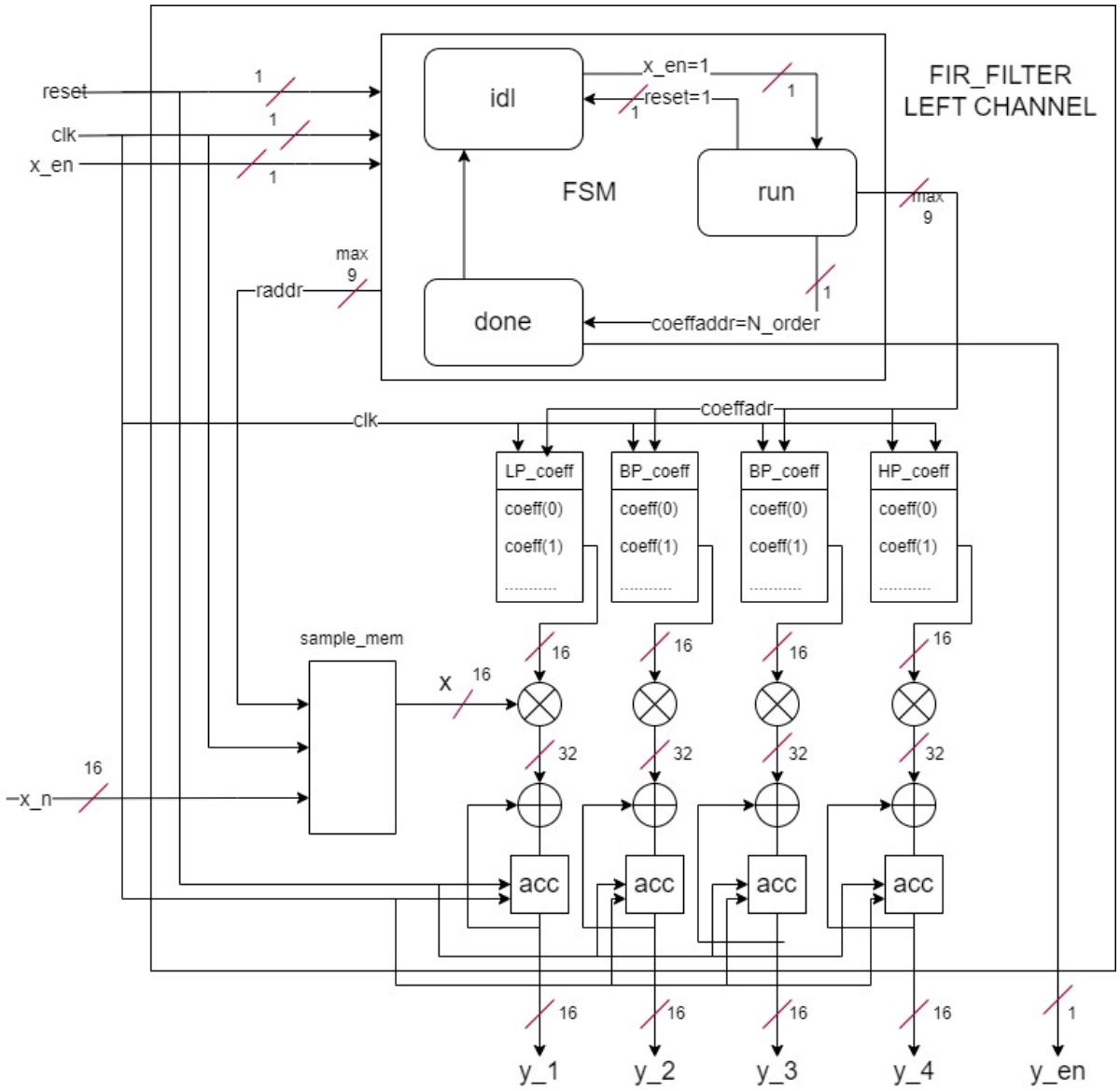

Figure 4: filter block diagram for the left channel. The numbers beside the red lines indicate the bits used for the signals

The filter banks as seen in the figures above for the left respective right channel are identical except that the enable signal for the right channel is the inverted version of the enable signal for the left channel.

The state machine (FSM) has three states: *idl*, *run* and *done* and works as depicted in the figure below. When in state=*idl* and *x_en* becomes 1 the next state is run and the *raddr* is set to *waddr*-1 and *coeffaddr* to 0. The state will be run until *coeffaddr=N_order*. When state=*done*, *y_en* becomes 1 and gives the signal to the next module that the outputs *y1*, *y2*, *y3*, *y4* are ready. If the reset signal is 0 then state will be *idl* and *waddr*, *raddr* and *coeffaddr* will equal to 0 next clock cycle.

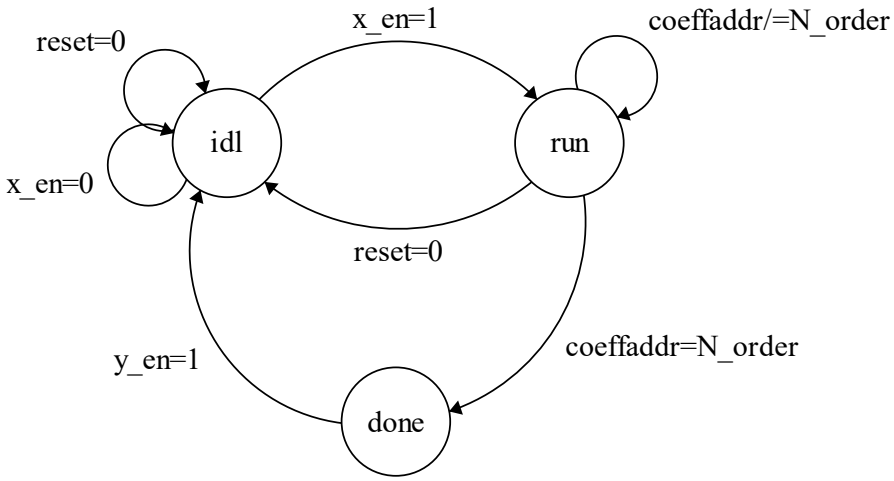

Figure 5: State machine for the filter banks

Simulation

The figure below depicts four impulse responses when testing the filter bank with a testbench in the simulation program Modelsim. The result shows that we have one low-pass filter, two band-pass filters and one high pass-filter.

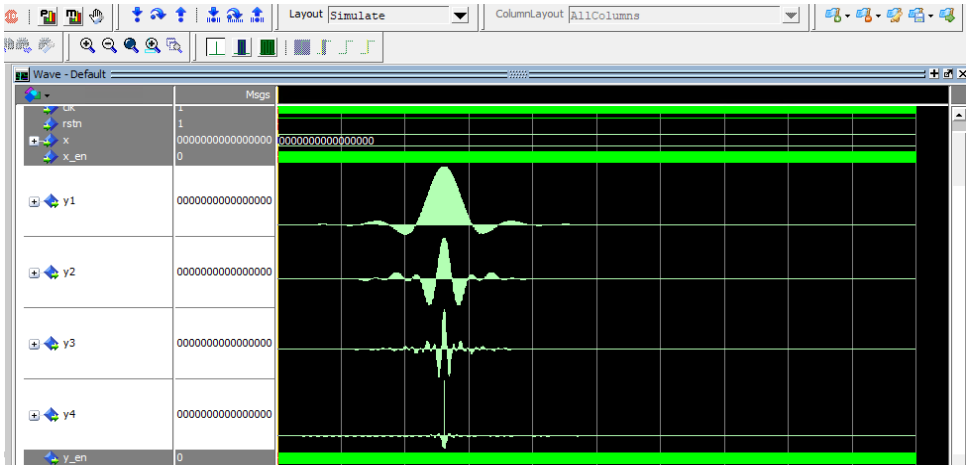

Figure 6: simulation result of testing the filter

Power Calculator

We want to know the power of the samples (for the right and the left channel), and to have this power on a logarithmic scale to represent them in a good way. For that we use two programs, the first does the calculation of the power and the other does the logarithmic scales (the second also sends a part of the power to improve the global print).

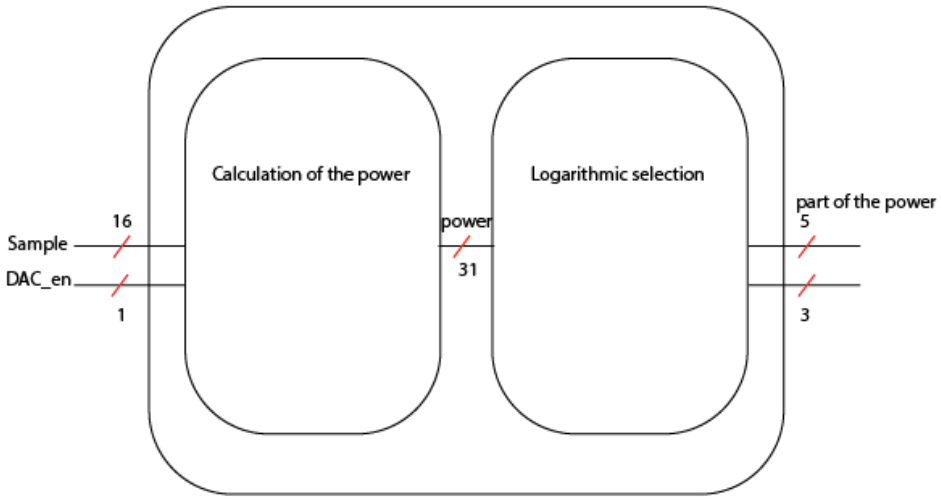

Figure 7: Diagram of the Power calculator

For the calculation of the power, we continuously sum up the sample levels and multiply the sum with 1-H in order to give weight to the recent samples. We need to find a good H (close to 0), for the test we take 1/4096. The purpose is to adapt this formula:

$$P = \sum_{n=0}^{\infty} x[n] \cdot (1-H)^n$$

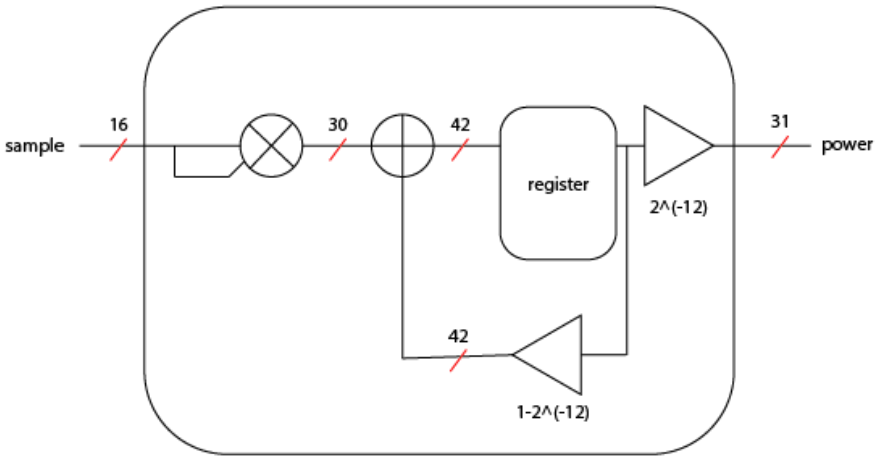

Figure 8: Diagram of the calculation of the power

For the logarithmic scale we use a state machine because we just want to know where the first bit at '1' in "power" is located. For that we save "power" in a temporary signal, and we shift it at each clock signal until its first bit is a '1'. After we send the number of this bit and the three bits which follow.

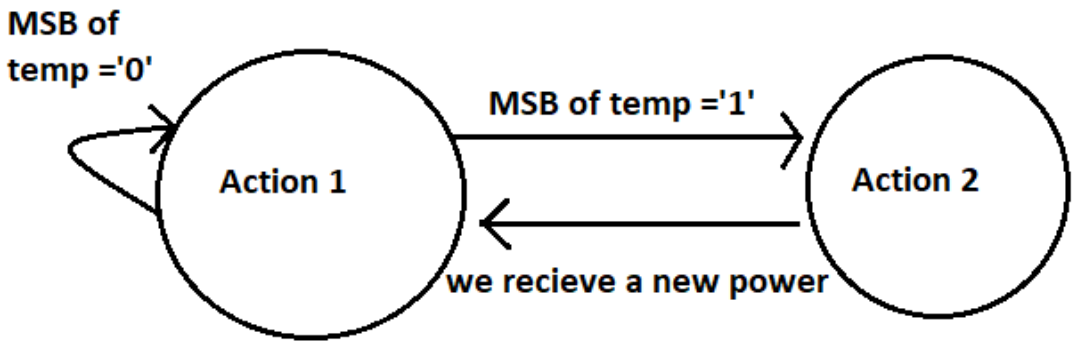

Figure 9: Representation of the state machine to do the logarithmic scale.

With temp a temporary signal, which takes the value of "power" when it changes.

During Action 1: we shift of one bit to left temps, and we increment a counter

During Action 2: we send the three first bits of temp after his MSB (35 to 33), and we send the counter

Oscilloscope

We will print the oscilloscope on the background of the screen at the same time as the foreground. So, we will need to print line by line, in fact to print a curve we only need to print one dot by line: In other word we need to know the *Hcount* where to print for each *Vcnt*'s value.

We do that in two parts, the first one will take the samples and *Vcnt* and return the *Hcount*, the other part will just print the dot in the same times as the other part of the screen. The first part also needs to transform the sample's value to get a logarithmic scale.

For that the first part is also divided in two, one part will create the logarithmic scale. The other will register them and after, return the good one in consideration of the *Vcnt*. The logarithmic scale will be created like in the power calculator. The register will just use an index, and we will move the index while we save the sample's value, after it will just return the good value (at index - Vcnt).

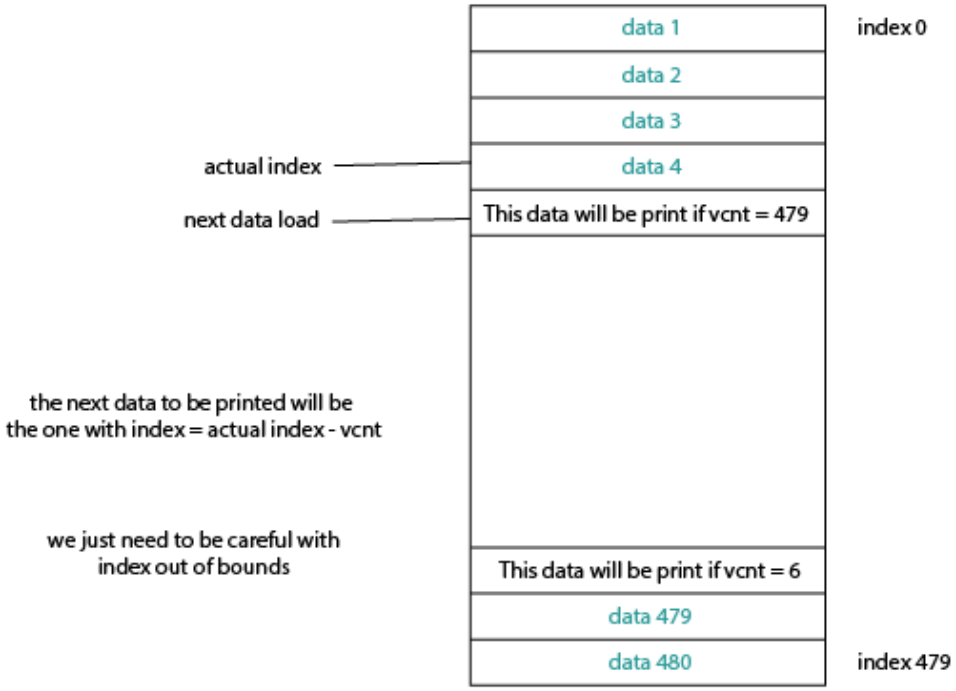

Figure 10: Register logic.

VGA Logic

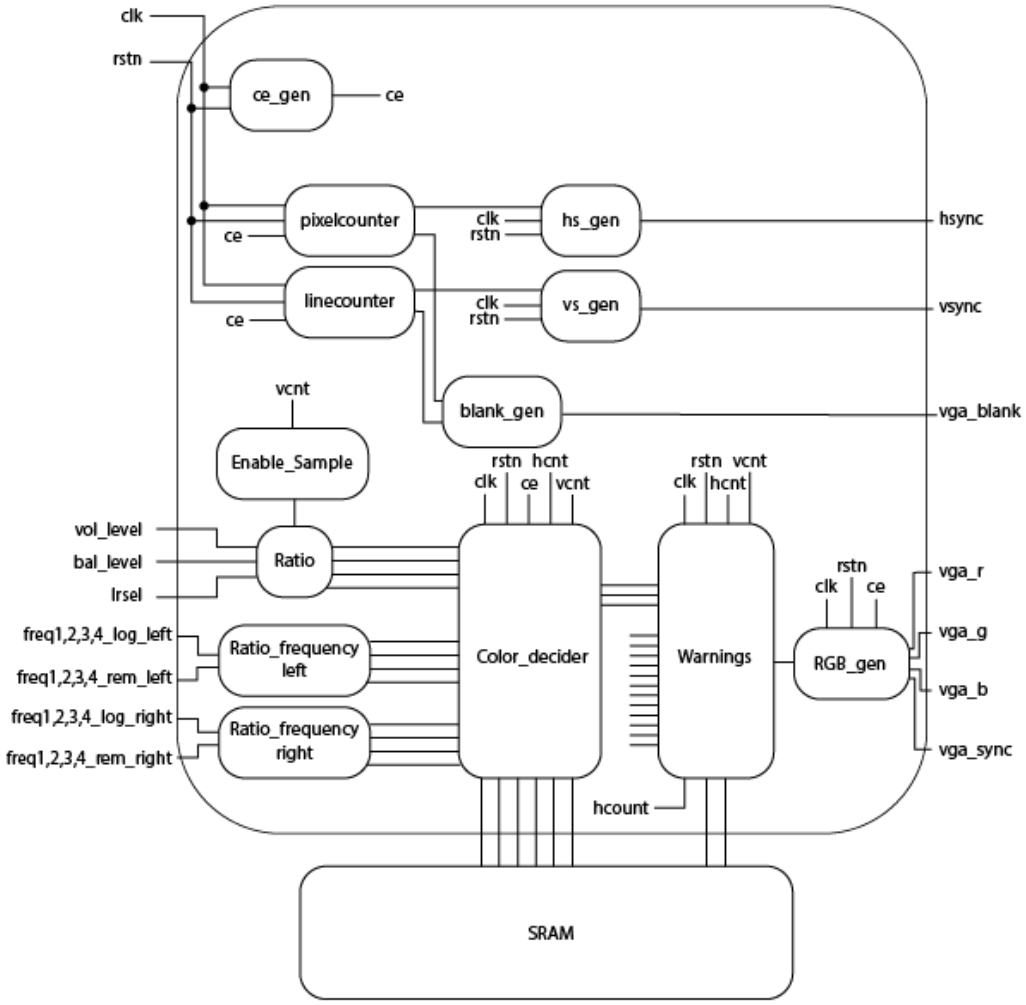

Figure 11: VGA_Control block diagram

The VGA control module is composed of two distinct parts. The first one manages the control signals for the VGA like *hsync*, *vsync* and *vga_blank*. This part is the combination of the module *ce_gen*, *pixelcounter*, *linecounter*, *hs_gen*, *vs_gen* and *blank_gen*. All those modules have the same implementation as in Lab3.

The second part controls the color of the pixels printed on the screen.

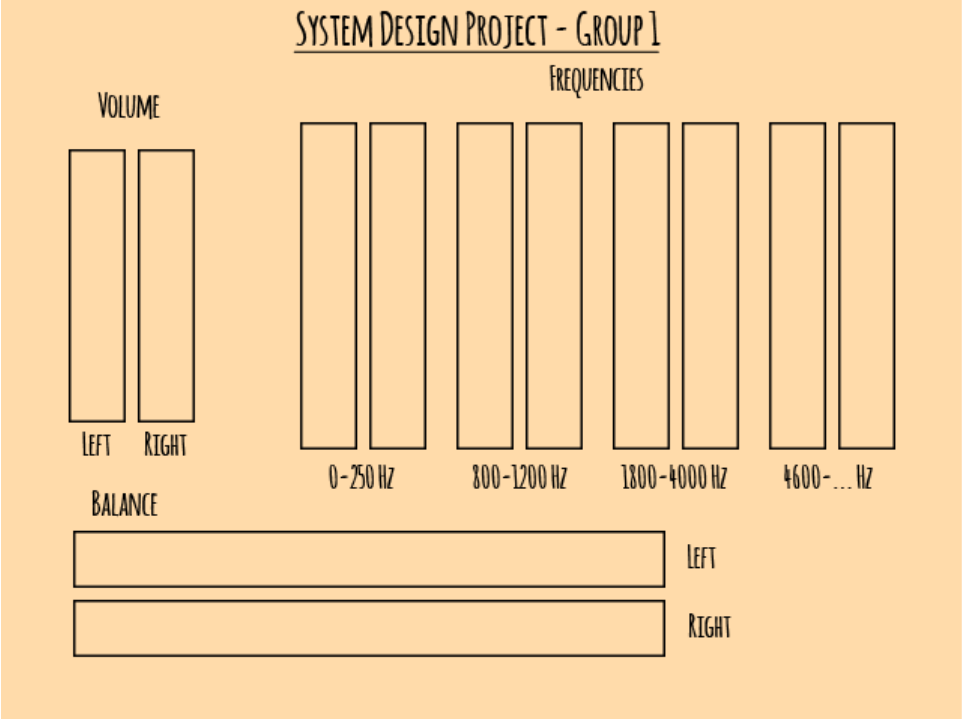

Figure 12: Background picture on the VGA screen

To decide the color of each pixel different modules are used. First, the VGA_Control module receives the volume and the balance level as well as the power of each signal generated with the filters. This one is divided into two signals, the first one is the power of two of the most significant digits and the second is the three following bits. To use those values on the VGA screen, the ratio modules convert power and level values into a line are column values. In fact, it calculates the limit between the colored area and the background area inside the gauges.

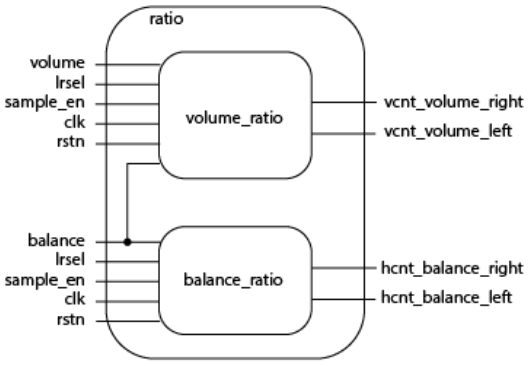

Figure 13: Ratio block diagram

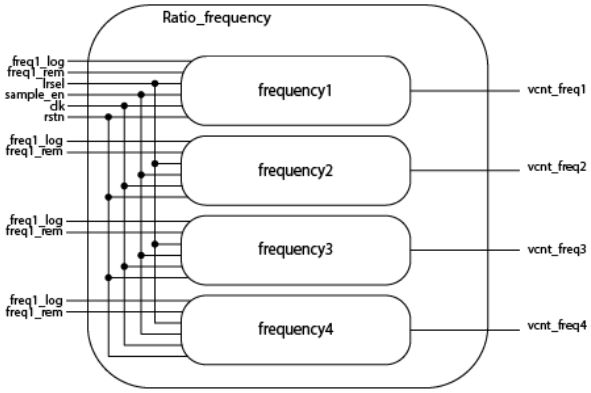

Figure 14: Ratio_frequency block diagram

In the ratio module, the input is simply multiplied by a coefficient and an offset is added to match with the background picture. This coefficient corresponds to the number of pixels each step is equal to.

In the ratio_frequency module, the scale is logarithmic, then the power of two of the most significant digits is multiplied by a first coefficient and added to the remaining. Moreover, as in the first case an offset is added. The first coefficient corresponds to the number of pixels between the values 2i and 2^{i+1}.

Nevertheless, the input values change more often than the screen refresh. Therefore, we implement an enable_sample module to permit to get new values when the screen does not print pixels, in other words, when *vga_blank* is on. This module generates an enabled signal used inside the ratio modules.

Ones the limits are calculated it is necessary to print the gauges on the screen. This is the functionality of color_decider module. To simplify this module is a large if statement which describes each area where the gauge must be printed. In addition, it generates the control signals and the address to use the SRAM, where the background picture is stored. A first *pixcode* value is sent, with the color of the gauge or with a default color and a background enable signal. In conclusion, this module prints the gauges and decides when it is necessary to get the pixel color inside the SRAM.

All this information is sent to the Warnings module. It takes the last decision about the color of the pixel. In fact, with this module some warnings and a mute icon can be printed thanks to another if statement and two ROM memories containing the drawing of the warning and the mute icon. Also, the background is loaded in this module. To summaries, with an if statement the module decides if it is necessary to print a warning or the mute signal and where it must be printed, and if no icons have to be printed then it chooses to display the background. Finally, outside the process a WHEN...ELSE statement permits to write over *pixcode* if it is the position of the oscilloscope.

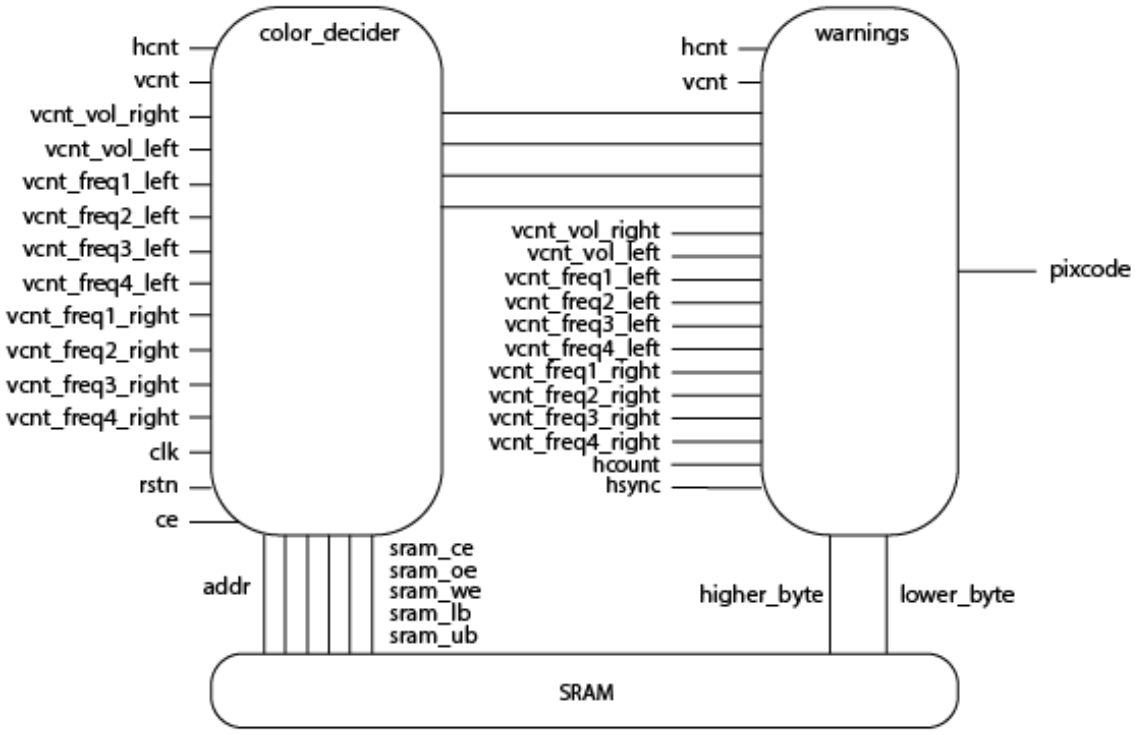

Figure 15: Color_decider and Warnings blocks diagram

To finish, the *pixcode* value is sent to the RGB_gen module which traduces it into a RGB format.

Challenges

One challenge was how to cooperate efficiently because the group members had different schedules. We tried to discuss and plan our project in both the physical lab room and online via teams.

Experiences

We had the opportunity of improving our collaboration skills which is an essential part of an engineer's job. In a project there are many things that can go wrong and not as planned. Thus, it is necessary to be prepared for such issues and find a solution to them as soon as possible.

Omid Najafi: I experienced that some problems just need time to be solved. In the beginning I thought that the designing part of the project would be the most difficult part and I think it turned out to be true for me. You need to know what you want to create before starting to write the code for it.

Nell Party: This project taught me the difficulties of a group project. In fact, one of the biggest difficulties in a group project is communication. It was deficient sometimes and slowed the work. For example, with the documents to give back. When we miss communication, it is difficult to finish it in time because each member does not know if the others finished. So, thanks to this project I know now that it is important to keep track of the advance of each member of the group and have good communication about each part of the project to do not to be blocked.

Henrik Lindgren: I learned that FPGA development tools such as Quartus, may not be best adapted for working cooperatively. You need to organize the project well to make the integration of all individual parts smooth and there are a lot of configuration and meta data files to keep track on apart from the pure source code.

Johan Klasén: Simulation testbenches are a very powerful and useful tool. However, they do take a lot of time to set up. If you get stuck on a problem for too long it is well worth using it to make some progress with the troubleshooting. If you have the possibility to use and adjust the testbenches form the lab series, that should probably save some time. And if you are planning to work on a big module where you know you will be making several gradual changes, it might be a good idea to create a testbench very early on. I've also been reminded how unforgiving low-level programming languages can be, where a single character typo, or a '1' instead of a '0' can lead to hours of unnecessary troubleshooting.

Apollinaire Criquet: I have learned the importance of writing some design specifications before doing any code. That is to make sure that everybody has clearly understood what he needs to do and what the other will do (but no one must understand how the others will do their part). I also learn that communication is very important, I meet some trouble with that. For example, when it has been the time regroup all the part, I meet some difficulty knowing where the developments of the other part are, and how we can regroup them.

Improvements

One of the possible improvements would be to add some key to modify the oscilloscope. For example, we could augment or reduce the speed of the oscilloscope or in the same way change the scale of it. This would need to modify the PS2 decoder and the oscilloscope part.

Another improvement which looks impossible or at least difficult is to add the possibility which the keyboard to modify the range of the filter. This will also need to modify the PS2 decoder and the filter, and maybe to implement the Fourier transformation to do the filter. But this requires having some good knowledge of the Fourier transform and will need a lot of time to calculate each filter output for just one sample. After all of that, add the possibility to modify the Fourier transformation.

It will be possible to add other background, and to modify the background while the consumer uses the project. This will need to set up other background and to find a way to change the background during the utilization and that in a smooth way.

User Manual

This product has seven functionalities, which are:

- The possibility for the user to listen to music, podcast, radio...
- Change the volume on the speakers
- Change the balance of the speakers
- Mute the speakers
- Have access to frequency analyzes
- Get an overview of the signal shape
- Be notified when the sound is too high

To enjoy this product nothing is easier.

On the keyboard, the up arrow increases the volume, the down arrow decreases it, the right arrow shifts the balance on the right and the left arrow shifts the balance on the left. Finally, the M key activates or deactivates the mute functionality.

On the VGA screen, the following information are given: the level of the volume and the balance, the power of each frequency band, the oscilloscope is on the background, the warnings are provided at the top of each gauge and a mute icon appears when this functionality is activated.

Time summary report

Student	Task	Planned time (hour)	spend time (hour)
Apollinaire Criquet	Power calculator / Oscilloscope	90	85
Henrik Lindgren	Volume and Balance	104	49
Johan Klasén	Keyboard	104	64
Nell Party	VGA Control	112	102
Omid Najafi	Filter bank	112	112

Keyboard

The keyboard module should have been pretty quick to implement, but due to a small error a lot of extra time was spent do debug it.

Volume and Balance

Filter bank

The designing part of the filter bank took a little more time than the planned time. The time spend on writing VHDL code did not differ much from the planned time but debugging and optimizing the code needed almost 5 more hours in comparison to the planned time. On the other hand, fixing the last errors required less hours. There is no significant deviation between the spend and planned time for writing the report.

Power calculator

For the power calculator, it takes less time than expected to do the VHDL code, and it was not very complicated to implement. But the test and the adaptation to the other module has been the hard part. In fact, it has been hard to do the test without the screen.

Oscilloscope

This part takes more time than expected on the time planner, and this in the majority because it has been hard to implement it with the rest of the project. But to write and to design this part has not be so long, but it has been difficult on many tricky points.

VGA Control

For the VGA, some parts were longer than expected like the color decider module because of the adjustment of each pixel. On the contrary, the ratio modules were quick to implement. Finally, the main part of the VGA Control module was done in only two weeks, which provides time to implement the priorities 2, like the warnings or the oscilloscope, but also to fix the last esthetic issues.

Project files

The project files are placed in K:\TSIU03\Groups\Group_1\Only_Project on the university's computers.