Design Specification
Group 1

1 Table of Contents

2 Introduction

The project is based on sound analysis and sound modification. In fact, at the end of the project the user will be able to modify the volume and the balance of the input sound. Moreover, the frequency will be analyzed. All the information collected by the system will be printed on a VGA screen.

3 Block Diagram

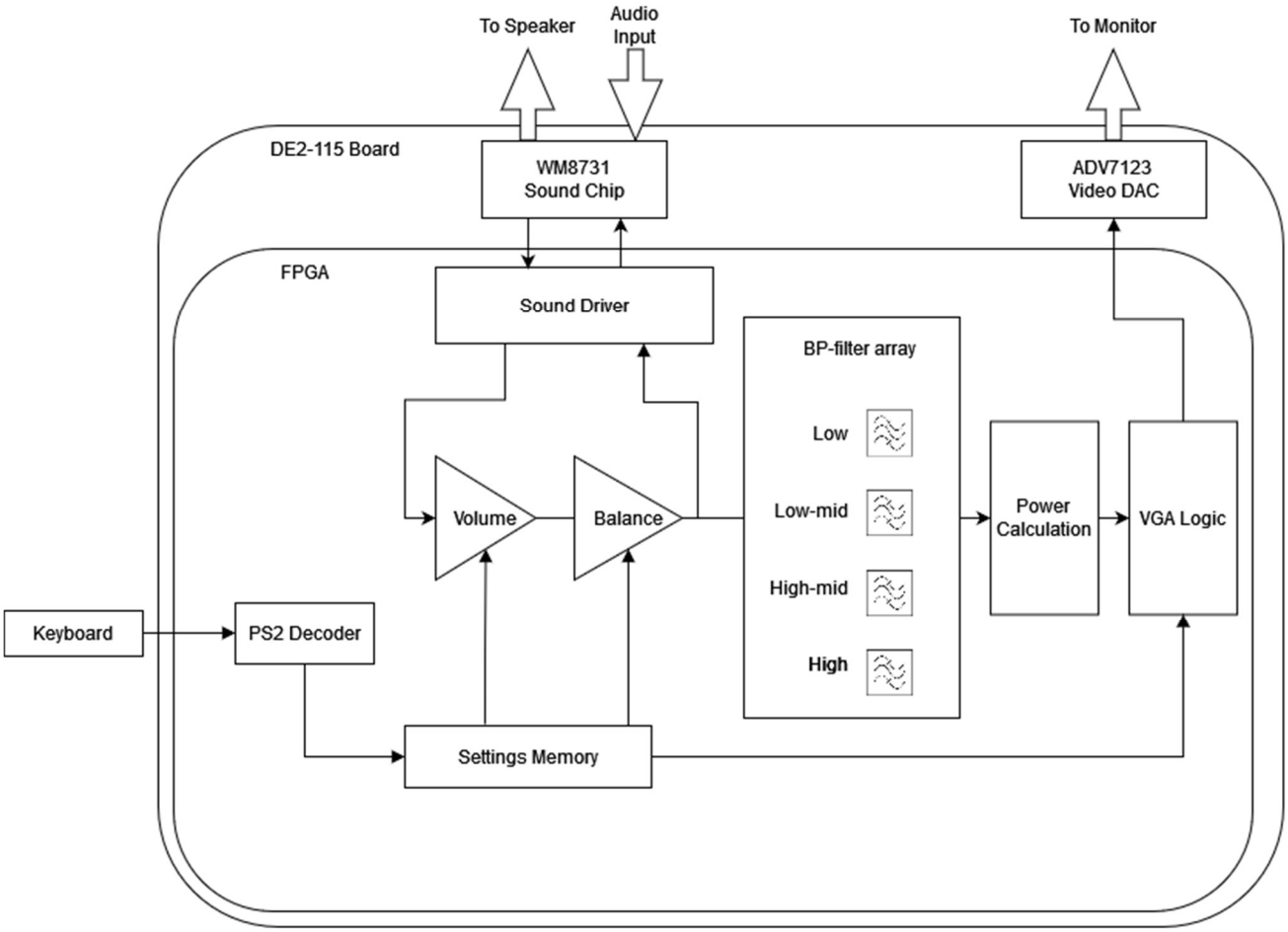

Figure 1: Overall system block diagram.

The system consists of several modules with their own respective functions. When the user presses a key on the keyboard, a signal is sent on the PS2 bus which is decoded by the PS2 Decoder module. If the key matches one which should change a setting, the settings memory is modified. The volume and balance settings are directly read by the volume and balance modules through which the samples from the sound driver are fed and modified. The adjusted signals are sent back out to the sound driver again while also going to the Band Pass-filter array. The filter array divides the signal into bands and their power levels are calculated in the Power Calculation module. These levels are used by the VGA Logic module to render indicator bars.

4 Block description

In this section the subblocks will be described.

4.1 PS2-Decoder

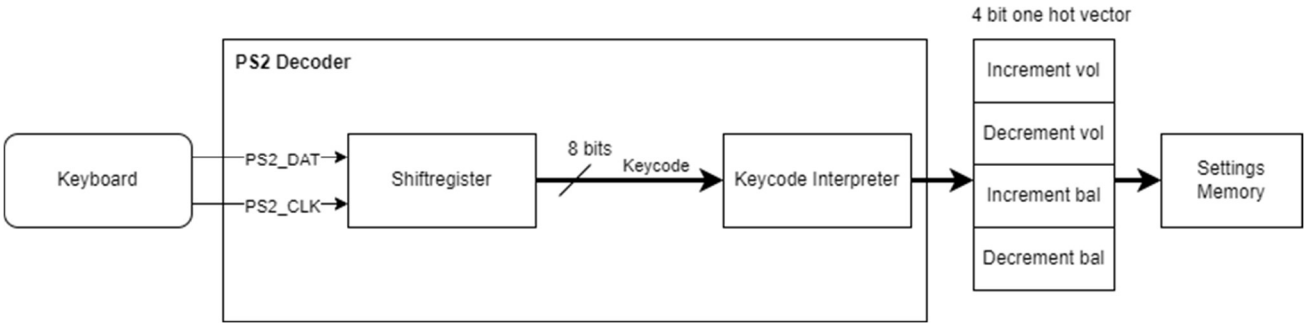

Figure 2: PS2-Decoder.

4.2 Sound Driver

Use the Lab4 code to configure the Sound Driver and input and output sound.

Firstly, a shift register is used to store the serial data from the keyboard as full 8-bit values representing the sent keycodes.
The keycode values are passed on to a keyboard interpreter which looks for expected keycodes for the volume and balance adjustments.
The interpreter uses a 4 bit "one hot bit vector" as an output signal, setting a single bit to 1 corresponding to an expected change. This is either an increment or decrement of the volume or balance, which in turn is passed on to the settings memory where the value of the volume or balance is adjusted as long as it does not put the values out of the predetermined range.
The range for the sound settings is 0-10 for volume and 1-9 for balance, as such a 4-bit value is needed for the output signals "vol_level" and "bal_level".

4.3 Volume and balance

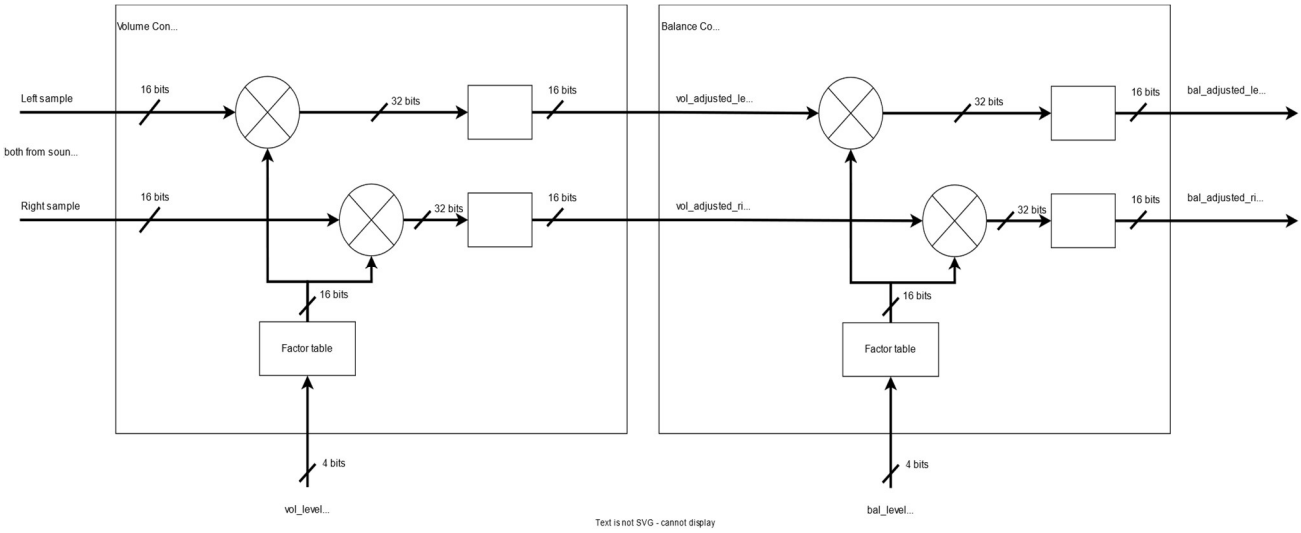

Figure 3: Volume and balance control.

We increase/decrease the volume of the signal by factor corresponding to at most 4 dB for each step. The level is converted into a factor which is multiplied by the sample value. For the balance control we will use different multipliers for the left and right channels. At the neutral level, both channels will be at full level (multiplied by 1.0). A level below neutral corresponds to leftward panning and will result in a decrease in volume of the right channel, whereas a level above neutral will result in a decrease on the left channel.

4.4 Filter bank

The incoming samples are stored in the internal RAM of the FPGA. Each filter bank 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 MATLAB which provides the coefficients needed for filtering of the signal. The coefficients also are stored in the internal ROM of the FPGA.

An FIR filter is defined by:

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

• x[n] is the input signal.

• y[n] is the output signal.

• hk is the kth filter coefficient.

• N is the length of the filter.

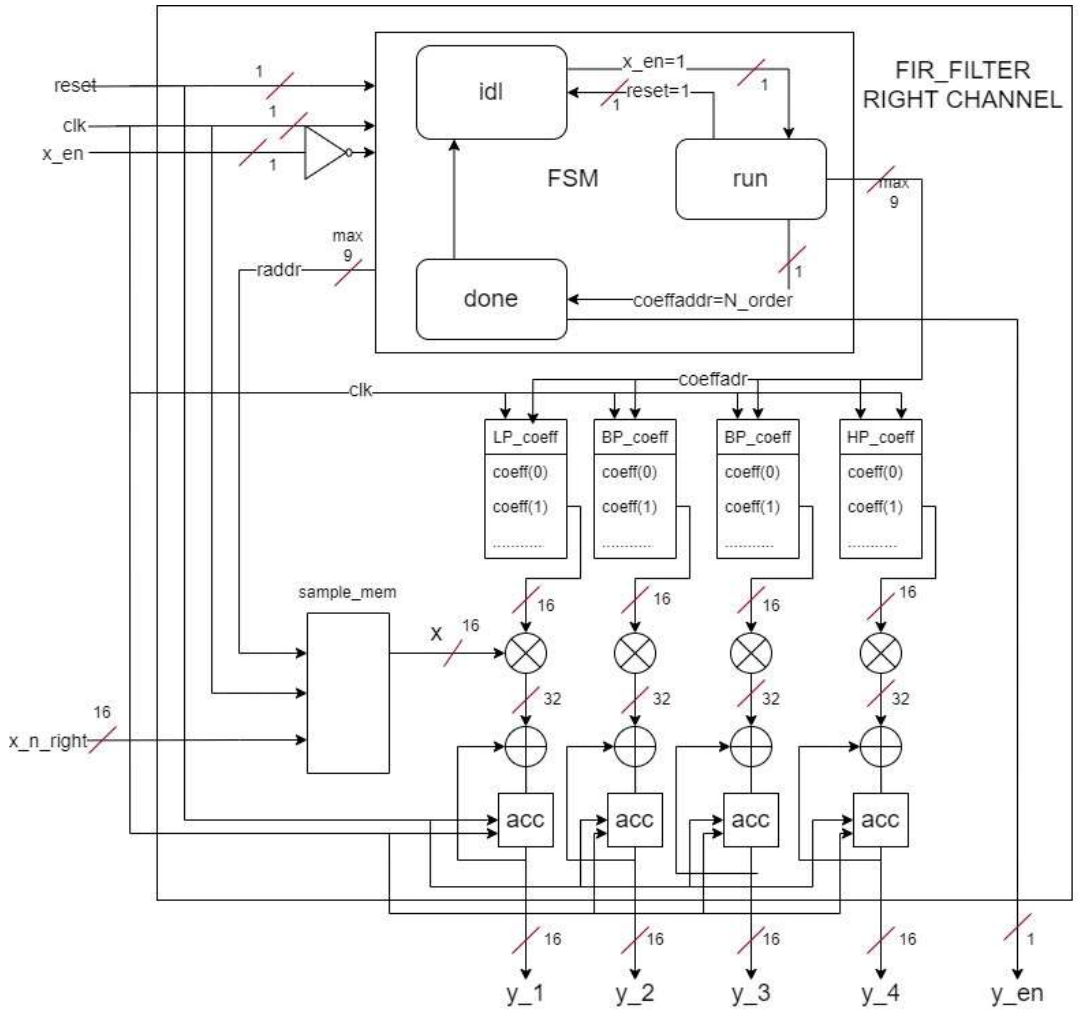
FIR_FILTER
RIGHT CHANNEL
idl
FSM
run
done
reset
clk
x_en
raddr
max
9
x_en=1
reset=1
coeffaddr=N_order
coeffadr
LP_coeff
BP_coeff
BP_coeff
HP_coeff
coeff(0)
coeff(1)
sample_mem
x
x_n_right
16
32
acc
y_1
y_2
y_3
y_4
y_en

Figure 4:Filter block diagram for the right channel.

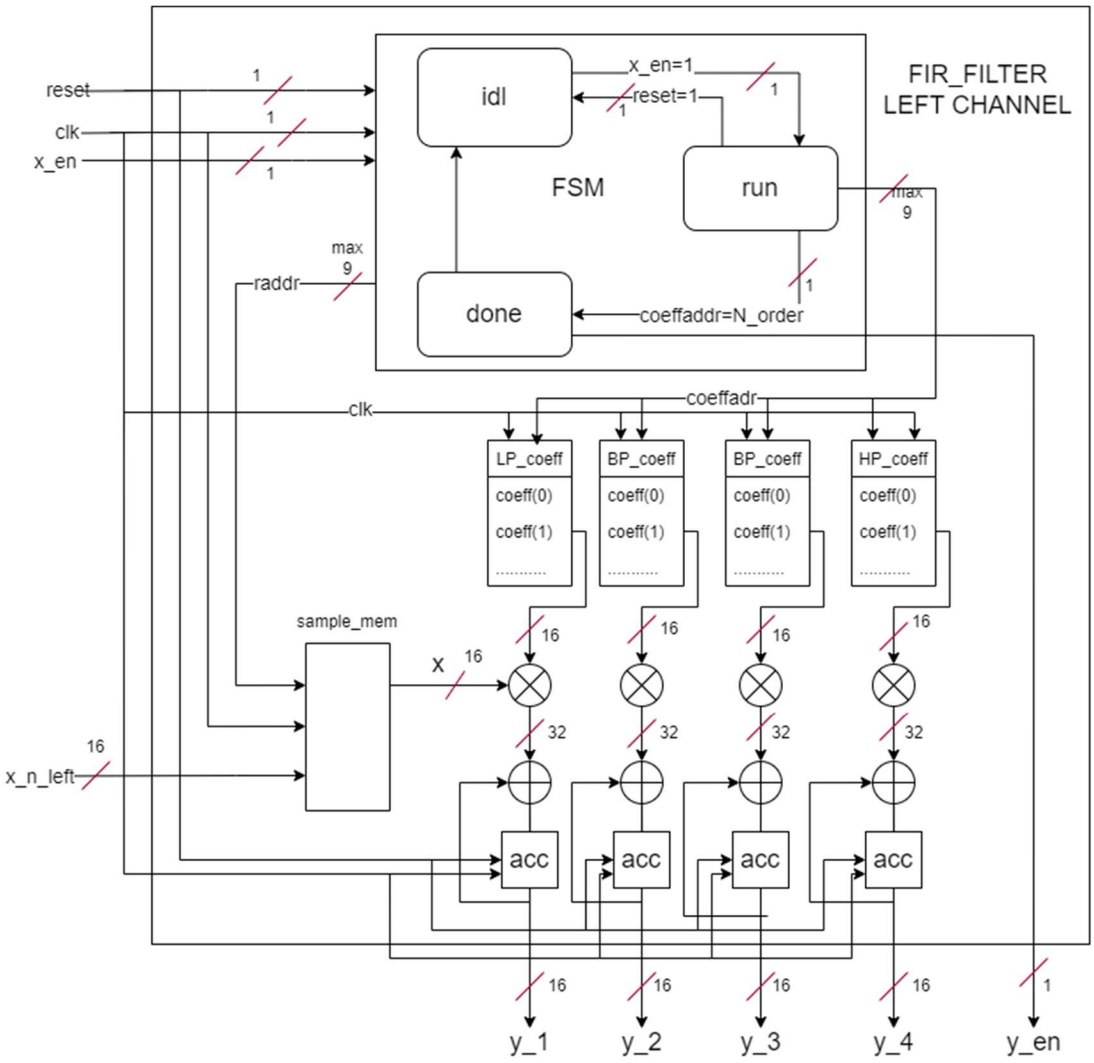

Figure 5: Filter block diagram for the left channel.

The filter banks as seen in the figures above for the left respective right channel are identical except that the enabled signal for the right channel is the inverted version of the enabled 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 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 next module that the outputs y1, y2, y3, y4 are ready.

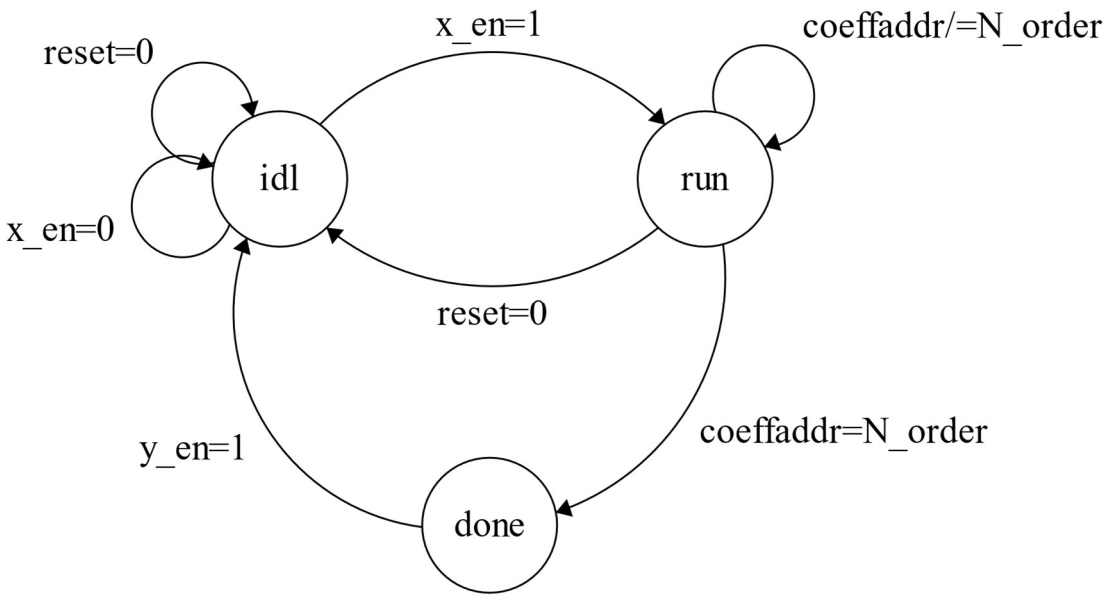

Figure 6: State machine for filter bank

4.5 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).

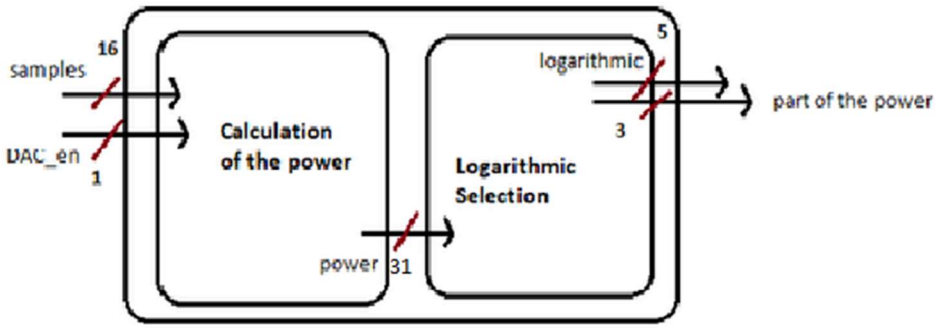

Figure 7: Diagram of the Power calculator.

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 1/32. The purpose is to adapt this formula $P = \sum_{n=0}^{\infty} x[n]^2(1-h)$

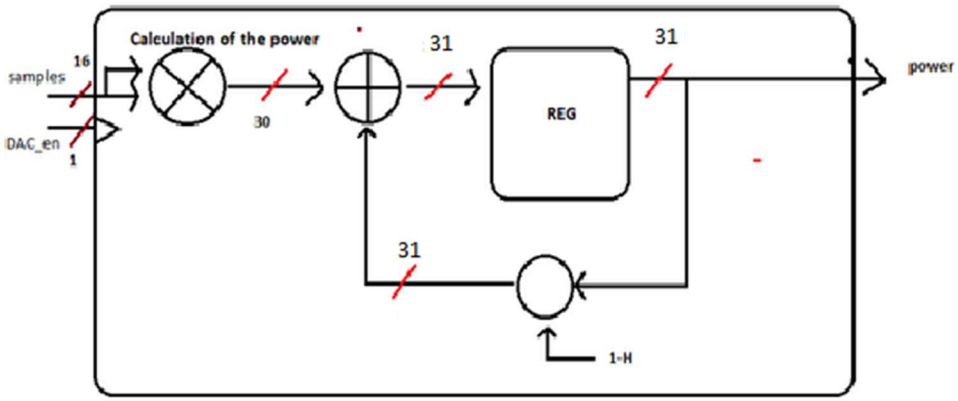

Figure 8: Diagram of the calculation of the power

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.

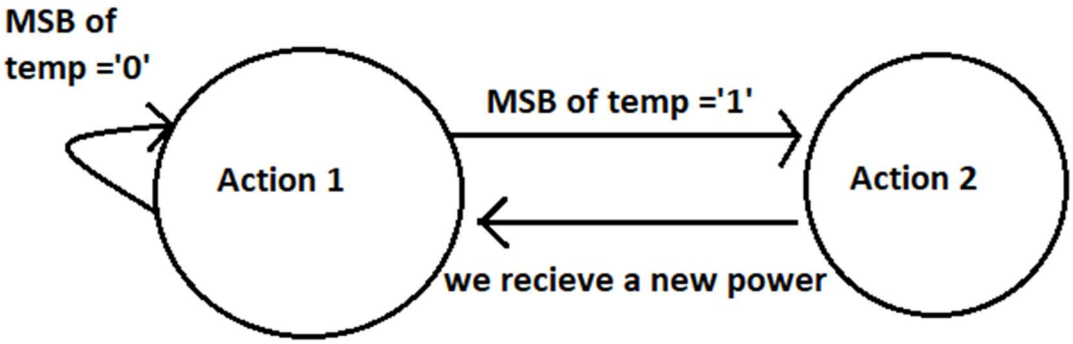

Figure 9: Representation of the state machine to do the logarithmic scale

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,

we increment a counter

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

we send the counter

4.6 VGA Logic

VGA screen block diagram:

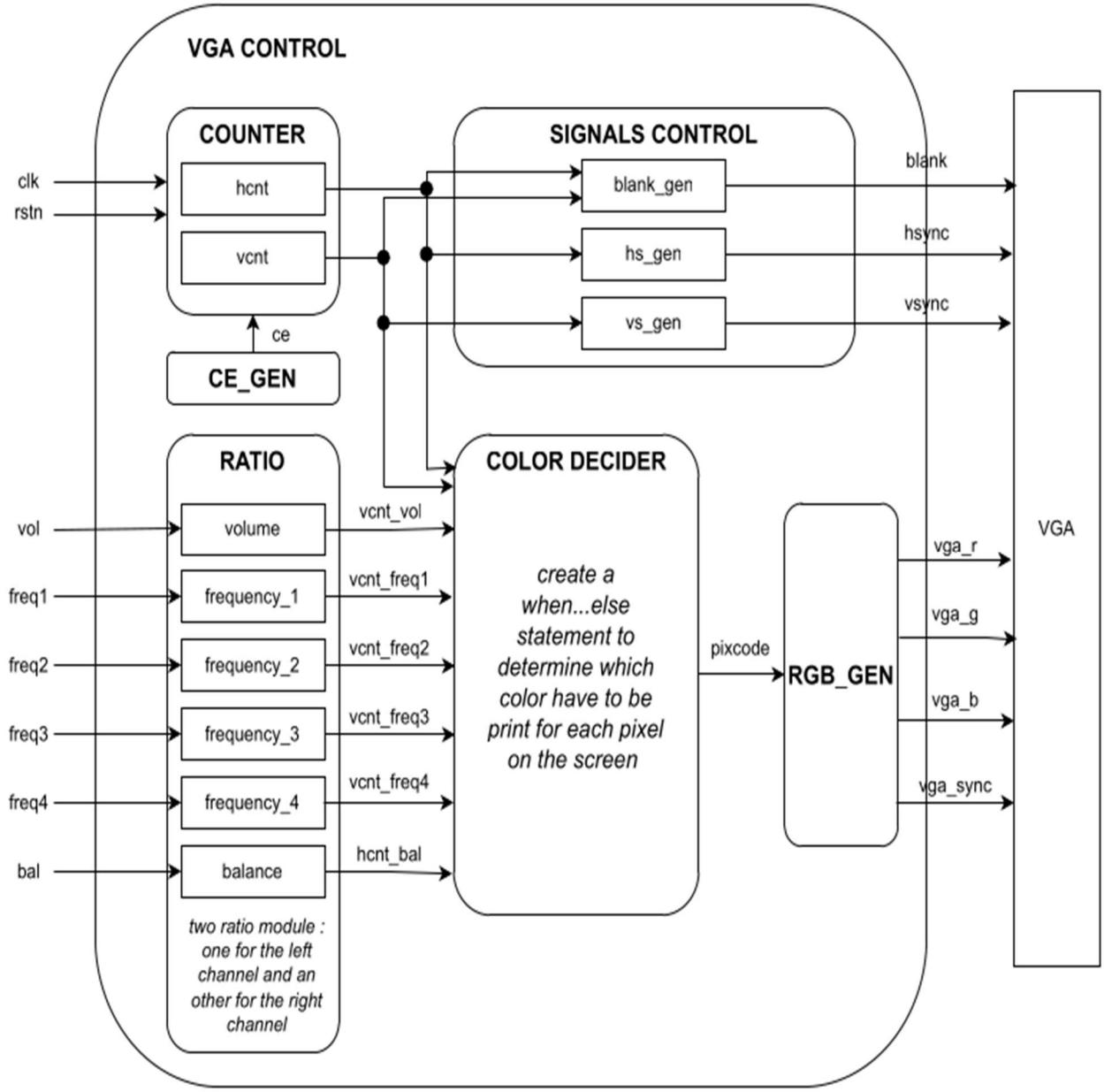

Figure 10: VGA Display Logic

4.6.1 Inputs

Name of the variable	Size	Description	Origin	Destination
ENABLE SIGNALS				
lrsel	1 bit	*Right/Left selector*	Power Calculator	Ratio
INDICATOR SIGNALS				
Volume	4 bits	*Level of the volume [1;10]*	Settings	Ratio
Balance	4 bits	*Level of the balance [0;9]*	Settings	Ratio
F1_log	5 bits	*Most significant digit*	Power Calculator	Ratio
F1_remaining	3 bits	*Following bits of log*	Power Calculator	Ratio
F2_log	5 bits	*Most significant digit*	Power Calculator	Ratio
F2_remaining	3 bits	*Following bits of log*	Power Calculator	Ratio
F3_log	5 bits	*Most significant digit*	Power Calculator	Ratio
F3_remaining	3 bits	*Following bits of log*	Power Calculator	Ratio
F4_log	5 bits	*Most significant digit*	Power Calculator	Ratio
F4_remaining	3 bits	*Following bits of log*	Power Calculator	Ratio

4.6.2 Intermediary signals

Name of the variable	Size	Description	Origin	Destination
hcnt	10 bits	*Column counter*	Counter	Blank_gen hs_gen color_decider
vcnt	10 bits	*Line counter*	Counter	Blank_gen vs_gen color_decider
pixcode	8 bits	*Color code of the next pixel*	Color Decider	RGB_gen
Sample_en	1 bit	*Indicate when to load new values*	Counter	Ratio
vcnt_volume_left	10 bits	*line where the gauge has to stop*	Ratio	Color Decider

vcnt_freq1_left	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq2_left	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq3_left	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq4_left	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
hcnt_balance_left	10 bits	*column where the gauge has to stop*	Ratio	Color Decider
vcnt_volume_right	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq1_right	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq2_right	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq3_right	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
vcnt_freq4_right	10 bits	*line where the gauge has to stop*	Ratio	Color Decider
hcnt_balance_right	10 bits	*column where the gauge has to stop*	Ratio	Color Decider

4.6.3 Outputs

Name of the variable	Size	Description	Origin
blank	1 bit	*out of the screen*	Blank_gen
hs_sync	1 bit	*Finish each line*	Hs_gen
vs_sync	1 bit	*Finish each column*	Vs_gen
VGA_r	8 bits	*Red value*	RGB_gen
VGA_g	8 bits	*Green value*	RGB_gen
VGA_b	8 bits	*Blue value*	RGB_gen
VGA_sync	1 bit	*End of the frame*	RGB_gen

4.6.4 Module description

COUNTER: generate the signals hcnt and vcnt as implemented in lab 3, as well as sample_en which pulses for a single clock cycle right before vcnt wraps around to zero.

SIGNALS CONTROL: generate the control signals (blank, vsync, hsync) as

implemented in lab 3.

CE_GEN: generate the clock enable signal (ce) as implemented in lab 3.

RGB_GEN: get the pixcode signal from COLOR DECIDER and generate vga_r, vga_g, vga_b and vga_sync as implemented in lab 3.

RATIO: get the value of the power of each element evaluated (volume, balance, frequency band) and return the line or the column where the gauge hast to stop. There are two kinds of scale. The first one is an arithmetic scale and the other one is a logarithmic scale.

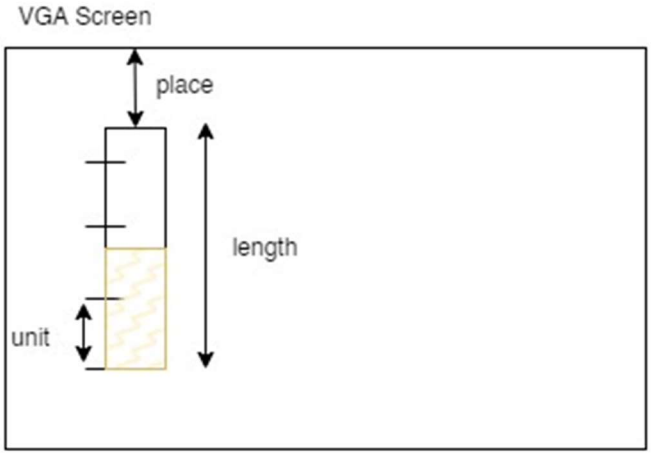

Figure 12: Example of an arithmetic as drawn on the VGA screen.

- *place*: number of lines/columns between the top/left of the VGA screen and the left top corner of the gauge.
- *length*: height of the gauge in pixels
- *unit*: number of pixels by unit (determine what is one unit for each element for example 3dB = 50 pixels)
- $output = place + (length - input \times unit)$
- Choose the unit so as to use shift and not multiplier

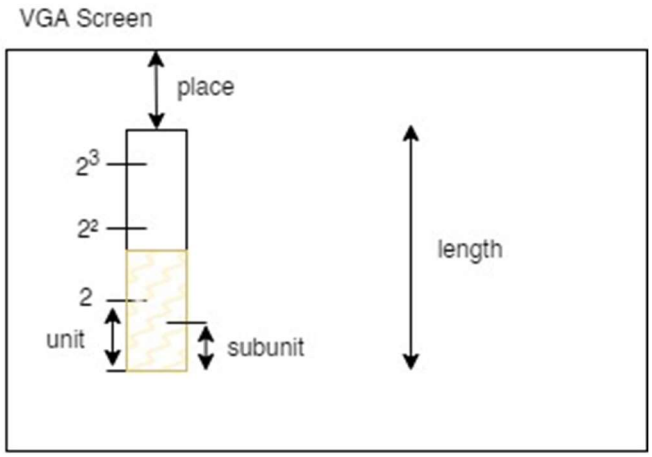

Figure 13S: Example of arithmetic as drawn on the VGA screen.

- *unit*: number of pixels between 2i and 2i+1
- *subunit*: same definition as the unit in arithmetic scale part but the scale is determined by unit

submodule	unit	subunit	length	place
Volume	4 dB = 17 pixels		187 pixels	100 pixels
Balance	12,5% = 50 pix-els		400 pixels	45 pixels
Frequency 1	14 pixels	2 pixels	224 pixels	100 pixels
Frequency 2	14 pixels	2 pixels	224 pixels	100 pixels
Frequency 3	14 pixels	2 pixels	224 pixels	100 pixels
Frequency 4	14 pixels	2 pixels	224 pixels	100 pixels

Volume unit = length / 11

Balance unit = length / 8

Frequency unit = length / 16

Frequency subunit = Frequency unit / 7 = length / 11

$output = cntr \times unit + remaining \times subunit$

RATIO modules must be duplicated twice, one for right channel and another one for left channel.

COLOR DECIDER: when…else statement which decide which color is apply at each pixel position. The decision is taken thanks to hcnt and vcnt variables but also with vcnt_vol, vcnt_freq1, vcnt_freq2, vcnt_freq3, vcnt_freq4, hcnt_bal (from right and left channels)

When the pixel does not correspond to a variable area get the information in ROM memory.

Condition on	**Hcnt (with thickness)**	**Vcnt (with thickness)**	**Hcnt (without thickness)**	**Vcnt (without thickness)**
Vol_left	45,82	100,285	55,77	105,280
Vol_right	91,128	100,285	96,123	105,280
F1_left	199,236	100,285	204,231	105,280
F1_right	245,282	100,285	250,277	105,280
F2_left	303,340	100,285	308,335	105,280
F2_right	349,386	100,285	354,381	105,280
F3_left	407,444	100,285	412,439	105,280
F3_right	453,490	100,285	458,485	105,280
F4_left	511,548	100,285	516,543	105,280
F4_right	557,594	100,285	562,589	105,280
Bal_left	45,444	356,393	50,439	361,388
Bal_right	45,444	402,439	50,439	407,433

Values of the conditions in the if statement

4.6.5 Timing

The timing problem is information to print on the screen will arrive continually and not necessarily at the same time but also not when we wanted to refresh the screen (every 837 900 system clock cycles).

To solve this problem, an enabled signal can control the Radio module. In other words, when the enable signal is equal to '1' then the Ratio module updates the values of the output, else the output conserves their values.

4.7 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 Vcount's value.

We do that in two parts, the first one will take the samples and Vcount 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 Vcount. 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 - Vcount).

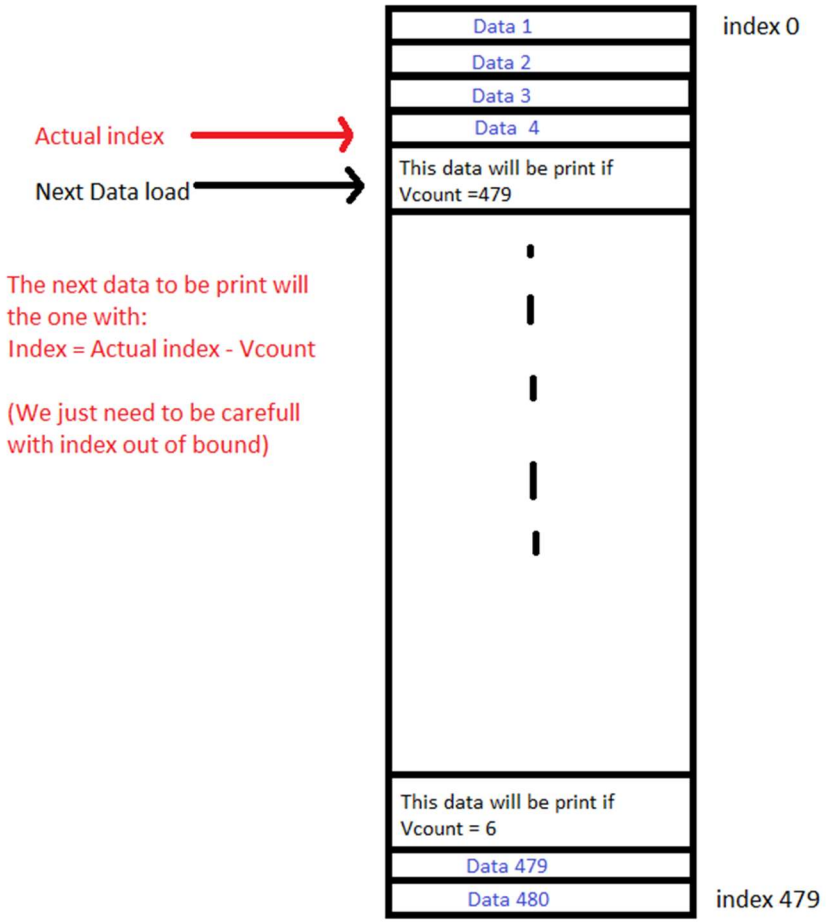

Figure 10: Register logic

4.8 Warnings:

The objective is to add some warnings at the top of each gauge when the level exceeds a value. We decided to create an ROM block with the picture of the warning inside, and as the Color decider part creates an if statement to place the warnings at the right place. Some added conditions permit us to decide when the warning must be printed on the screen.

5 Communication

PS2 => Settings: the last key press (10 bits, 8 for the word of the key, and 1 for E0 and F0) or the order (4 bits) to modify the settings.

Settings => Vol and Balance: the current level of each one (9 levels for balance, 11 for volume) (4 bits for both).

Sound driver => Vol => Balance => Filter: the sound sample modified by Vol and Balance (16 bits), ADC_en and DAC_en (1 bit).

Filter => detection of the power: the sound samples (16 bits) to calculate power of, and a DAC_en (1 bit).

Detection of the power => VGA: the current power of the signal after each filter (need to be logarithmic, 5 bits for the most significant digit and 3 bits for the remaining).

Oscilloscope => VGA: the position of the sample on the current line (10 bits)

Settings => VGA: the current level of the Vol and the Balance (9 levels for balance, 11 for volume) (4 bits for both).

6 Challenges

Some challenges are

- Pipeline the circuit
- Use proper good word lengths
- Implement the band-pass filters
- Combine all the modules to build a bigger system
- Implement the state machines

The fact that the modules will be written by different people in the group makes it more challenging. The group members have different schedules which can cause that they can't meet to cooperate at any time.