Hardware Implementation of Automatic Gain Controller for Active Hearing Protectors

Niklas Aldén

Department of Electrical and Information Technology
Lund University

Advisor: Joachim Rodrigues and Nedelko Grbic

December 8, 2015

Printed in Sweden
E-huset, Lund, 2015

Abstract

This thesis focus on implementing an automatic gain controller (AGC) in hardware for usage in active hearing protectors. The proposed solution matches and improves on both the dampening and audio quality compared to an equivalent commercial product. The listening experience for the user is improved by applying an equal attenuation for both left and right ear.

The AGC works by estimating the decibel level of an audio sample and applying an appropriate gain. In case the noise level is so high that the users hearing might be damaged, the sample is attenuated to a harmless level. Otherwise, the sample is outputted without any dampening. The proposed solution is verified on an FPGA and prepared for fabrication of an ASIC in 65nm CMOS technology.

The aim has been to optimized the algorithm to result in an integrated circuit with a small area and low power consumption. By implementing a time multiplexed resource sharing algorithm, the circuit area is reduced with 28%. Together with the use of voltage scaling, the simulated energy dissipation per clock cycle is reduced with 86%.

Acknowledgments

I would like to thank my advisers Dr. Joachim Rodrigues and Dr. Nedelko Grbic: First, for proposing this thesis; secondly, for their guidance and support during this period. Also a thanks to Ph.D. Student Oskar Andersson for his help with the tools. Finally, a special thanks to my family for always supporting me during my studies.

Niklas Aldén

December, 2015

Table of Contents

List of Figures

List of Tables

Abbreviations

AC'97 . . . Audio Codec 1997
ADC . . . . Analog to Digital Converter
AGC . . . . Automatic Gain Controller
ASIC . . . Application Specific Integrated Circuit
ASMD . . Algorithmic State Machine with Data path
CMOS . . . Complementary Metal-Oxide-Semiconductor
DAC . . . . Digital to Analog Converter
FIR . . . . . Finite Impulse Response
FPGA . . . Field Programmable Gate Array
FSM . . . . Finite State Machine
I/O . . . . . Input and Output
IIR . . . . . Infinite Impulse Response
LPHVT . . Low Power, High Voltage Threshold
LPLVT . . . Low Power, Low Voltage Threshold
LPSVT . . Low Power, Standard Voltage Threshold
LSB . . . . Least Significant Bit
LUT . . . . Lookup Table
MSB . . . . Most Significant Bit
MUX . . . Multiplexer
PCM . . . . Pulse-Code Modulation
PnR . . . . Place and Route
VHDL . . . Very high speed integrated circuit Hardware Description Language

Chapter 1

Introduction

This chapter will introduce the background and the objectives of this thesis. The outline of the report with a short description of each chapter will also be provided.

1.1 Motivation

According to the Swedish Work Environment Authority, people working in an environment with an equivalent noise level between 75dB and 80dB, during an 8 hour work day, are entitled to have hearing protectors provided for them. If the noise level reaches 85dB or more, there is a risk of permanent hearing damage so the employer should wear hearing protectors. [1]

There are two different types of hearing protectors: passive and active. The passive type, like earplugs for example, are the most common ones. They work by dampening all sounds before it reaches the eardrum, regardless of the surrounding noise level. In situations where communication is important, but with a chance of high noise levels, a pair of active hearing protectors could be a better fit over the passive kind. Active hearing protectors uses an Automatic Gain Controller (AGC) to adjust the sound volume, depending on the noise level, before the sound reaches the users ears. This means that harmless sound is not damped[1] whilst dangerous sound levels are attenuated to a safe level.

Currently available active hearing protectors are either fully analog, fully digital, or mix between analog and digital. It is motivated to have digital hearing protectors for the simplicity of adding extra features in the future, but additional functions usually means worse battery life. The digital parts are implemented on Digital Signal Processors (DSPs), which are more efficient than having general purpose processors doing the calculations. However, compared to a purpose-build Application Specific Integrated Circuit (ASIC), the DSP is physically much larger and consumes far more energy.

[1] Some implementations amplifies low sound levels in order for the user to hear better.

1.2 Thesis Objective

This thesis will be a further development on the work done in [2], which focused on digitizing an existing analog solution. Their conclusions for suitable filter implementations will be used in this thesis, but where they programmed a floating-point DSP for verification, the result in this thesis will be written in VHDL and verified on an Field-Programmable Gate Array (FPGA) using fixed-point arithmetic.

The goal is to match and improve on the dampening capabilities and sound quality against a pair of commercially available hearing protectors, while using much less power. The hearing protectors used for comparison adjusts the dampening separately for each ear, independent of the other. This is good for locating the precise direction of a noise source far away. However, a strong impulse sound from one side, and soon after one from the other side, can cause a swaying sound effect when one ear hears the sound before the other ear and adjust the attenuation differently. This swaying effect is caused by a large difference in dampening for each ear, meaning one ear can pick up background noise while it is very quiet for other ear. When this difference changes rapidly, it could be a bit unpleasant for the user, and will therefore be solved in this thesis. By applying an equal attenuation for both ears, determined by the side with the most powerful noise, the swaying effect can be eliminated but still keeping the ability to locate the direction of a noise source.

The final design will be prepared for manufacturing of an ASIC in 65nm CMOS technology from STMicroelectronics, with focus on as low power consumption as possible.

1.3 Thesis Outline

Chapter 2 explains the basics of digital number representation and the introduced errors when converting analog signals to digital. The concept of the decibel scale and its usage in this thesis is also provided.

Chapter 3 describes the work of developing an algorithm in Matlab that filters and dampens the audio samples.

Chapter 4 describes the process of taking the algorithm, adjusting it, and translating it into hardware using a hardware description language, VHDL.

Chapter 5 goes through the optimizations done to the initial hardware implementation in order to make it more efficient. An in-depth description of the new algorithm is also provided.

Chapter 6 describes the additional interface and controller that is necessary for having the algorithm running on the FPGA.

Chapter 7 covers the implementation of the algorithm on an ASIC. The workflow of going from a hardware behavioral model to an integrated circuit is briefly described.

Chapter 8 presents the setup and the equipment used for verification, along with the results.

Chapter 9 contains the conclusions and suggestions for future work and further development.

Chapter 2

Theory

This chapter explains the basics on digital number representation, the source of rounding errors, and the usage of the decibel scale in this thesis.

2.1 Floating Point Representation

When handling decimal numbers in a computer, most of the times they are represented using floating point notation [3]. They are often called `float` in most common programming languages. These decimal numbers are represented as binary numbers using normalized scientific notation, meaning there is only one non-zero number to the left of the binary point[1], i.e. any leading bits that are zero are moved and instead used for representing the numbers fraction, thus increasing the precision of the number. This means that the position of the binary point is not fixed, it is *floating*, thereof the name floating point. The bit at position n to the right of the binary point has the weight of 2^{-n} [4].

For example, the number $^1/_2$ in base 10 is written in base 2 floating point as: $1.00\ldots \cdot 2^{-1}$, not as $0.10\ldots \cdot 2^0$

The advantage of floating point is the range of numbers that can be represented, from small fractions with high precision up to very large numbers. In general, floating point numbers are written as in (2.1) and the distribution of bits used for the exponent and fraction parts for the most common types, single and double precision, are shown in Table 2.1 and Table 2.2.

$$(-1)^S \cdot (1.F) \cdot 2^E. \tag{2.1}$$

where: S = Sign bit, 1 for negative, 0 for positive
F = Fraction bits
E = Exponent bits

[1] Binary point is the base 2 equivalent of the decimal point in base 10.

TABLE 2.1: IEEE 754-1985, 32-bit single precision floating point.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	Exponent								Fraction																						
1 bit	8 bits								23 bits																						

TABLE 2.2: IEEE 754-1985, 64-bit double precision floating point.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
S	Exponent											Fraction																			
1 bit	11 bits											20 bits																			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fraction continue																															
32 bits																															

Floating point notation offers great precision for a wide range of numbers, but calculating with them is difficult, time consuming and requires special arithmetic units. To avoid having to use specialized hardware for floating point calculations, fixed point numbers will be used in this thesis.

2.2 Fixed Point Representation

In contrast to floating point, the position of the binary point in fixed point notation is, as the name suggests, *fixed*. The user must in advance decide how many bits should be used for the integer part and for the fraction part. This is often written on so called *Qm.n*-format, where m is the number of integer-bits and n the number of fraction-bits. In this thesis, n will be 0, meaning all fixed point numbers will be interpreted as integers.

The fixed point representation is much easier to work with since the numbers are integers and much simpler hardware, such as regular adders, can be used. The disadvantage with fixed point is the limited range of numbers that can be represented. Floating point numbers adapts when the numbers increase or decrease, but with fixed point one is stuck with a predetermined range. [4]

If the most significant bit (MSB) were to be used for a sign bit and the remaining bits for the magnitude, one would end up with both a positive and a negative zero. To avoid having two definitions of the number zero, signed numbers are often represented in two's complement. In this representation, for an N-bit number the MSB has the weight of -2^{N-1} and the remaining k bits have a weight of 2^k, where $0 \leq k < N-1$. For example, the 4-bit integer -5 is written in two's complement as:

$$1011_2 = -2^3 + 2^1 + 2^0 = -8_{10} + 2_{10} + 1_{10} = -5_{10}. \tag{2.2}$$

With N bits, the minimum value of a number is -2^{N-1} and the maximum is $2^{N-1}-1$, meaning the range for a 16-bit number is:

$$\left\{-2^{15},\ 2^{15}-1\right\} = \{-32768,\ 32767\}. \tag{2.3}$$

A floating point number is converted into a fixed point number by multiplying it with 2^x, $x \in \mathbb{Z}^+$, and then rounded to the nearest integer. This introduces an error, because of the limited resolution, called quantization error. The resolution of a fixed point number is determined by the number of bits used, and can be seen as what impact the least significant bit (LSB) has in proportion to the whole number. For a large number, ± 1 do not make much of a difference, but for a small number it changes a lot. The difference in resolution when using various number of bits are shown in Table 2.3. Quantization error is discussed further in section 2.3.

TABLE 2.3: Comparison of fixed point resolutions. Different sizes of fixed point numbers and their minimum and maximum value, and the floating point equivalent of LSB.

	Range		
# bits	Minimum	Maximum	LSB resolution
4	-8	7	$1.25 \cdot 10^{-1}$
8	-128	127	$7.81 \cdot 10^{-3}$
12	-2048	2047	$4.88 \cdot 10^{-4}$
16	-32768	32767	$3.05 \cdot 10^{-5}$

2.3 Error Estimation and Bit Resolution

Error estimation is important because of the introduced error when converting floating point numbers to fixed point numbers. Similarly, there will always be a quantization error when converting an analog signal to a digital representation, because of the limited resolution depending on the number of bits used. A depiction of an analog-to-digital conversion and digital-to-analog recreation of the sampled signal is shown in Fig. 2.1.

An addition of two n-bit numbers can result in a $(n+1)$-bit sum and a multiplication of two n-bit numbers can result in a $(2 \cdot n)$-bit product [5]. This has to be taken into account. In order to not corrupt any data due to an arithmetic overflow, the size of the variable that stores the operands and the result needs to be well adapted. Few bits require less number of registers and smaller arithmetic units, which translates to less power usage and smaller chip area, but increases the round-off error. With more bits, the error is smaller but requires more registers and larger arithmetic units with larger chip area and higher power consumption

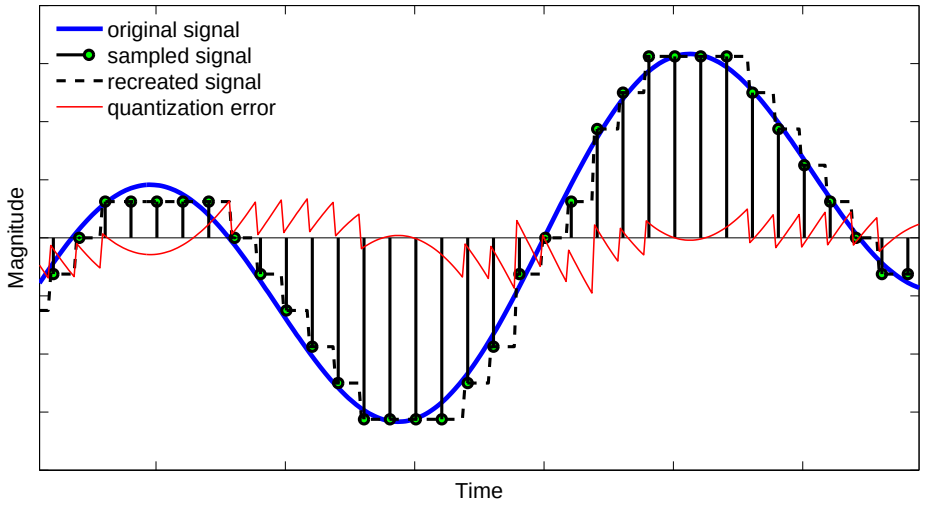


Fig. 2.1: Quantization error. Difference between recreated signal and original signal using 4-bit resolution, i.e. $2^4 = 16$ discrete digital values.

as a consequence. A trade-off is needed. The variance of the quantization error for different resolutions is shown in Fig. 2.2. As seen in Fig. 2.2, there is an exponential decrease in the variance, i.e. less deviation from the correct value, with more bits. At some point, the audio quality will be acceptable and adding more bits may not result in noticeable better audio quality. Different number of bits will be tested during the AGC algorithm development, but implementing in hardware, verifying, and comparing all different sizes is not in the scope for this thesis.

Matlab, which is used during development of the algorithm, supports both signed and unsigned integers with sizes of 8, 16, 32, and 64 bits. In each block of the design, only the number of bits needed to represent each signals largest absolute value is used. Every signal is studied to ensure that enough bits are used in every step to hold the signals value. In the Matlab implementation, one is limited to the aforementioned integer sizes but with a custom hardware design, any number of bits can be used, resulting in a more optimized design.

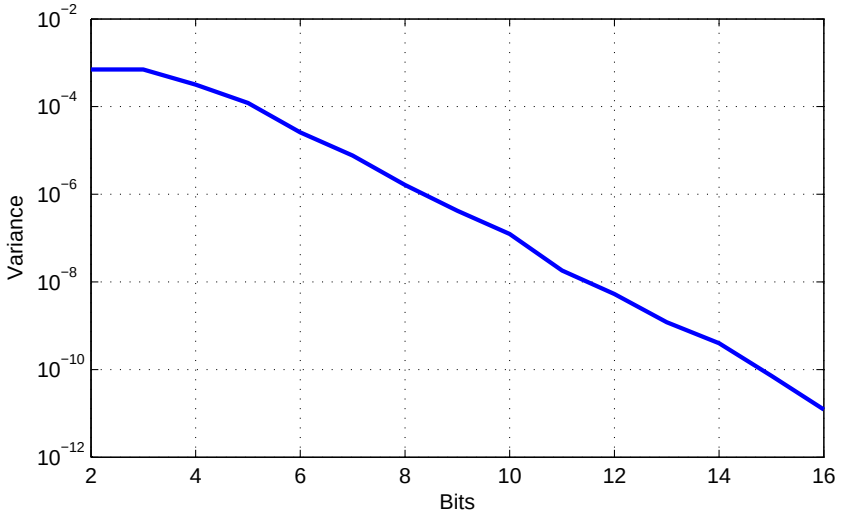


Fig. 2.2: Variance of quantization error for different bit resolutions.

2.4 dB

When measuring accurate noise levels, one tries to replicate the human auditory perception by applying a weighting filter usually called an A-filter, see Fig. 2.3. This is because the human ear has a different sensitivity for different frequencies. A sound with a frequency of a few kilohertz is heard clearer than a low- or high frequency sound. The measured noise is measured in dB(A) where the "(A)" denotes that the noise level is weighted with an A-filter [1].

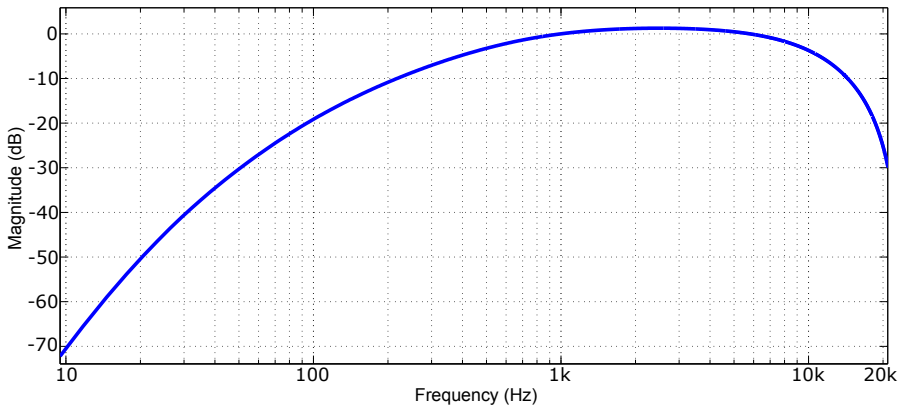


Fig. 2.3: Frequency response of A-weighting filter.

In this thesis, for simplicity the power of the sampled sound will not be weighted with an A-filter. Instead, the power of a sample will be the squared absolute value of what the analog to digital converter (ADC) samples and then transformed into decibel scale. This calculation is not the true A-weighted power, and therefore the lookup table (LUT) containing all the gain values has to have an offset to match the actual noise level.

When talking about sound levels in decibel, one refers to how much more power there is then the standard reference power, which is 10^{-12}W. In general, decibel is a measurement of how much one level differs from a specified reference level, see (2.4). Since decibel is a logarithmic scale, a negative value means the measured level is lower then the reference, a positive value means its more powerful, and zero means its at the same level as the reference value. The logarithmic scale also means that an increase of about 3dB, which may not sound like much at first, is actually a doubling in power: $10 \cdot \log_{10}(2) = 3.01029\ldots \approx 3$.

$$L_{\text{dB}} = 10 \cdot \log_{10}\left(\frac{P_1}{P_0}\right) = 10 \cdot \log_{10}(P_1) - 10 \cdot \log_{10}(P_0). \tag{2.4}$$

where: L_{dB} = Sound level in decibel
P_1 = Measured power
P_0 = Reference power

During development of the algorithm, there will be needs for calculating the power level from a given decibel level. When going back from a decibel value to the corresponding power level, assuming the reference power is zero, the following rewriting of (2.4) is made:

$$\begin{aligned} 10 \cdot \log_{10}(P) &= L_{\text{dB}} \\ \log_{10}(P) &= \frac{L_{\text{dB}}}{10} \\ 10^{\log_{10}(P)} &= 10^{\frac{L_{\text{dB}}}{10}} \\ P &= 10^{\frac{L_{\text{dB}}}{10}}. \end{aligned} \tag{2.5}$$

Chapter 3

AGC Algorithm Development

This chapter covers the development of the AGC algorithm using Matlab. An overview of the algorithm design is shown in Fig. 3.1. First an ADC samples the audio from a microphone and converts the analog signal to a binary fixed point number, see section 3.1. The sample then passes two filters described in section 3.2 before reaching the AGC (sections 3.3 and 3.4). When the sample is processed, and damped if necessary, it is converted back to an analog signal by a digital to analog converter (DAC).

A low latency for processing the audio samples is necessary. If there is a delay, even a small one, it will be annoying for the user. By processing one sample at the time, i.e. fetching one sample from the ADC, calculate and apply appropriate gain and then output the sample to the DAC before the ADC samples again. This means that the algorithm has to be faster then the sampling frequency of the ADC.

One important thing to remember during development is that everything has to work in hardware afterwards, meaning built-in Matlab functions cannot be used.

3.1 Fixed Point Conversion

The input samples during Matlab development comes from `.wav`-files which stores audio-samples in a range from -1 to 1, represented in floating point numbers. In the real world the samples are analog audio signals with infinite resolution. To make the calculations easier and faster, i.e. more efficient, the samples are transformed to fixed point two's complement integers.

Different number of bits, up to 16 bits, for the fixed point conversions are evaluated. The floating point samples are multiplied with $2^{\text{bits}-1}$ and then typecasted to a signed integer. In the end, there is no time to try out several different sizes in hardware. To not sacrifice too much on audio quality, 16 bits was chosen since this is the maximum resolution that the ADC on the FPGA can handle.

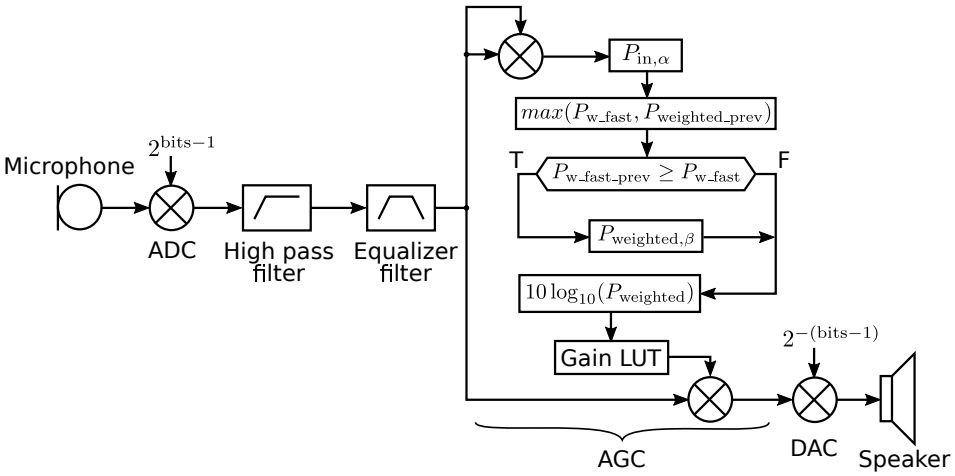


Fig. 3.1: Overview of algorithm implementation.

3.2 Filters

In [2] it was found that the most suitable filters would be infinite impulse response (IIR) filters instead of finite impulse response (FIR) filters. The IIR filters uses fewer components and have a shorter delay then a FIR filter with comparable frequency response, therefore IIR filters are more efficient. However, depending on the pole-placement of the filter coefficients, IIR filters can be unstable since the output is fed back. The filters used in this thesis are found to be stable.

3.2.1 High Pass Filter

The first filter is a high pass filter used for eliminating low frequency noise created for example by the wind. The frequency response of this filter is shown in Fig. 3.2. It is implemented in Direct-Form-I, Fig. 3.3, using (3.2).

The general equation for a first order IIR-filter is:

$$a_0 y(n) + a_1 y(n-1) = b_0 x(n) + b_1 x(n-1). \tag{3.1}$$

where: a_0, a_1, b_0, b_1 = filter coefficients, defined in [2]
$x(n)$ = current input sample
$x(n-1)$ = previous input sample
$y(n)$ = current output sample
$y(n-1)$ = previous output sample

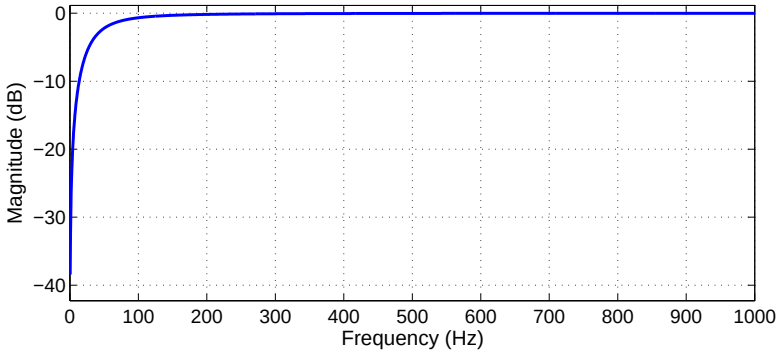


Fig. 3.2: Frequency response for high pass filter.

By dividing all coefficients in (3.1) with a_0 and rearranging we get:

$$y(n) = -a_1' y(n-1) + b_0' x(n) + b_1' x(n-1). \tag{3.2}$$

where: $a_1' = -0.9685$
$b_0' = 0.9842$
$b_1' = -0.9842$

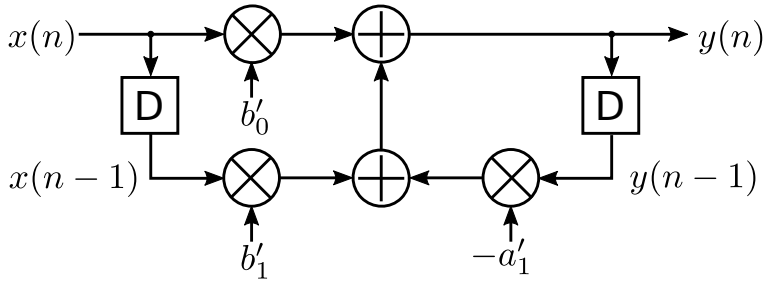


Fig. 3.3: First order IIR filter in Direct-Form-I.

Since the algorithm operates using fixed point arithmetic, the filter coefficients also has to be transformed into fixed point numbers. This is done by multiplying the coefficients with $2^{15} = 32768$, i.e. 16-bit fixed point. This multiplication will make the sample 32768 times larger after passing the filter, so the sample needs to be divided with the same amount after the filter.

This gives the fixed point coefficients:

$$\begin{aligned} a_1' &= -31736 \\ b_0' &= 32250 \\ b_1' &= -32250 \end{aligned}$$

3.2.2 Equalizer Filter

The equalizer filter limits the samples to a bandwidth of about 4kHz. The frequency response of this filter is shown in Fig. 3.4. It is implemented in Direct-Form-I, Fig. 3.5, using (3.4).

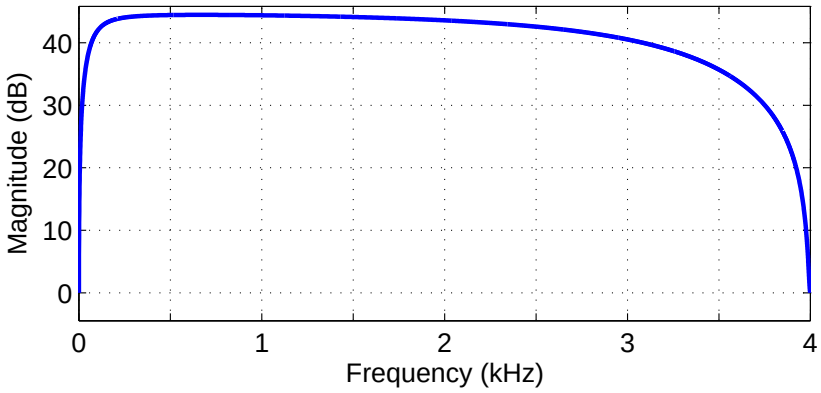


Fig. 3.4: Frequency response for equalizer filter.

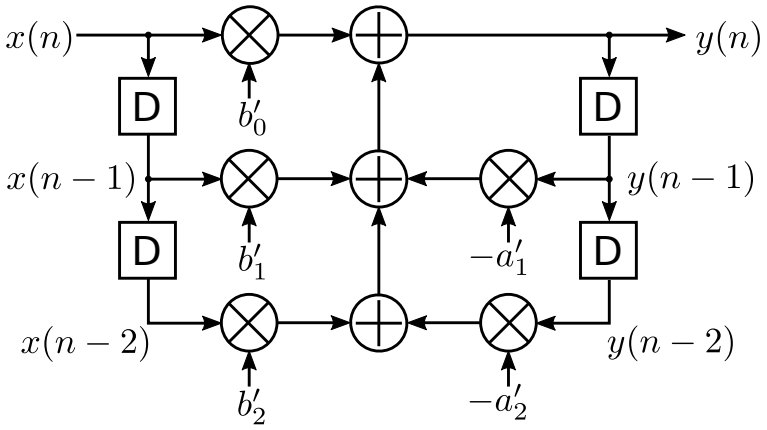


Fig. 3.5: Second order IIR filter in Direct-Form-I.

The general equation for a second order IIR-filter is:

$$a_0 y(n) + a_1 y(n-1) + a_2 y(n-2) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2). \quad (3.3)$$

By dividing all coefficients with a_0 and rearranging we get:

$$y(n) = -a'_1 y(n-1) - a'_2 y(n-2) + b'_0 x(n) + b'_1 x(n-1) + b'_2 x(n-2). \quad (3.4)$$

where: $a'_1 = -0.6108$
$a'_2 = -0.2947$
$b'_0 = 108.37$
$b'_1 = -0.6108$
$b'_2 = -107.66$

Converting to fixed point gives the filter coefficients:

$$\begin{aligned} a'_1 &= -20015 \\ a'_2 &= -9657 \\ b'_0 &= 3551068 \\ b'_1 &= -20015 \\ b'_2 &= -3527803 \end{aligned}$$

The equalizer filter with these filter coefficients amplifies the sample and in order to get an accurate power estimation, and not distort the sound, some dampening is needed. This is accomplished by dividing the sample after being filtered. However, division is a very complicated operation unless one can divide by two to the power of a positive integer, i.e. $sample/2^{bit} \Rightarrow$ shifting right *bit* number of times. The step response in Fig. 3.6 shows that the amplification caused by the filter is initially between 108 and 174 before steadily decreasing. The most suitable dampening factor will therefore be $2^7 = 128$, since $2^6 = 64$ and $2^8 = 256$ are either too small or too big. Seven rightshifts is not a perfect dampening, but the small difference in amplification afterwards will be acceptable.

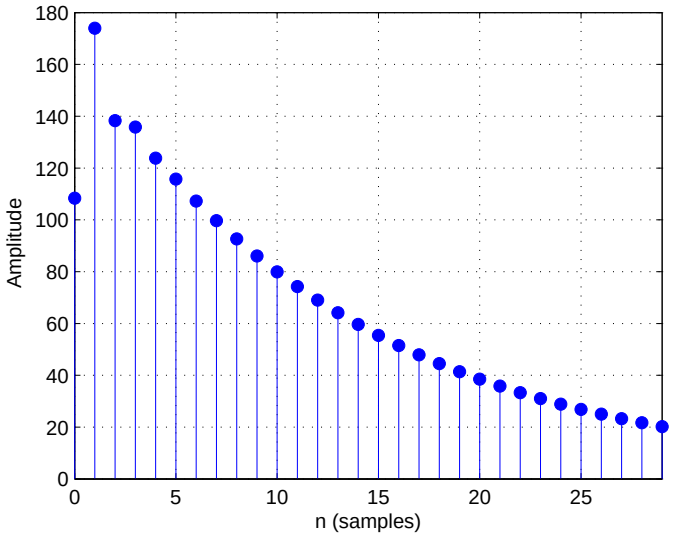


Fig. 3.6: Step response for the equalizer filter.

3.3 Signal Power Estimation

To limit the output to a harmless level, the power of the samples has to be estimated. If the output sound level is too high the hearing protectors will not fulfill their purpose and the users hearing might be damage. In [2] they used 82dB as the threshold limit, so 82dB will also be used in this thesis.

The constants α and β used from here on are parameters that determines the attack and release time respectively of the system, in other words how fast the system reacts to a change of intensity at the input. The aim is to have an attack time of less than 1ms and the release time should be around 300ms.

First, the power of a sample $x(n)$ is calculated by squaring its absolute value, i.e. $|x(n)|^2$. A fraction of the sample's power is then weighted with the time constant α against a part of the weighted power of the previous sample, as in (3.5),

$$P_{\text{w\_fast}}(n) = (1-\alpha)P_{\text{w\_fast}}(n-1) + \alpha \cdot |x(n)|^2. \tag{3.5}$$

The time constant α is used when the system needs to react fast to a change in power, typically when the power is increasing. The weighted power is stored in a signal called $P_{\text{w\_fast}}$, where the index denotes the fast reaction time.

The actual power that will be used when determining the gain factor for the AGC is a signal called P_{weighted}. In case the power calculated in (3.5), $P_{\text{w\_fast}}(n)$, is larger than the previously used power, $P_{\text{weighted}}(n-1)$, $P_{\text{weighted}}(n)$ is assigned the value of $P_{\text{w\_fast}}(n)$. Otherwise, the previous weighted power, $P_{\text{weighted}}(n-1)$

is assigned to $P_{\text{weighted}}(n)$. See (3.6). This is to ensure that the attenuation in the AGC is sufficient, i.e. the maximum calculated power has to be used.

$$P_{\text{weighted}}(n) = max\big(P_{\text{w\_fast}}(n),\ P_{\text{weighted}}(n-1)\big). \tag{3.6}$$

In the case of decreasing weighted input power, $P_{\text{w\_fast}}$, the current weighted power, $P_{\text{weighted}}(n)$, will be weighted against the time constant β, see (3.7), in order for the estimated power to descend slowly. If the power were to decrease too fast, the outputted audio would stutter during attenuation.

$$P_{\text{w\_fast}}(n) \leq P_{\text{w\_fast}}(n-1) \Rightarrow P_{\text{weighted}}(n) = (1-\beta)P_{\text{weighted}}(n). \tag{3.7}$$

When the weighted power to be used when determining the dampening is calculated, it is converted to its nearest decibel value. This rounding makes the lookup process easier since a smaller table is needed. Decibel is a ratio between a power value P_1 and another power value P_0, with P_0 as the reference, as defined in (2.4). The estimated weighted power of the sample $P_{\text{weighted}}(n)$ measured in dB is calculated as in (3.8) with 0dB as reference.

$$L_n = 10 \cdot \log_{10}\left(\frac{P_{\text{weighted}}(n)}{1}\right) = 10 \cdot \log_{10}\big(P_{\text{weighted}}(n)\big). \tag{3.8}$$

3.4 AGC and Gain Lookup Table

The AGC uses the weighted power of the current sample to find an appropriate gain in order to limit the output to a harmless noise level, a maximum of 82dB. If the sample's power is below 82dB, the gain applied is 1. Instead of evaluating a gain-function for each sample, a lookup table (LUT) with precalculated function results for each power level is more efficient. When a gain is found, it is multiplied with the input sample and the product makes the output sample.

The ideal values of the LUT are calculated as in (3.9), with a gain of 1 for any sample below the power limit, and an appropriate gain for a more powerful sample. See Fig. 3.7.

$$P_{\text{in}} \cdot G = P_{\text{out}} \leq 82\text{dB} \Rightarrow G = \begin{cases} 1, & P_{\text{in}} \leq 82\text{dB}; \\ \frac{82\text{dB}}{P_{\text{in}}}, & P_{\text{in}} > 82\text{dB}; \end{cases} \tag{3.9}$$

where: G = gain
P_{in} = power of input sample in dB
P_{out} = power of output sample in dB

One thing to remember is that the power of input samples are for convenience measured in decibel, but the sample is still a fixed point number in linear scale. This means that the gain must be calculated as in (3.10), i.e. using what was derived in (2.5), to have the correct value for samples over 82dB.

$$G = \frac{10^{82/10}}{10^{P_{\text{in}}/10}} \tag{3.10}$$

The ideal gain curve has a sharp edge where the gain starts to apply will have a bad effect on the sound quality. A smoother transition is desirable. Therefore, the values in the LUT near 82dB will be replaced with the functionvalues of a polynomial. This will make the transition at 82dB smoother and the sound quality will not be affected as much.

As previously discussed in section 2.4, P_{in} in (3.9) is not the true noise level since it is not A-weighted. During measurements in the anechoic chamber, section 8.2, it was discovered that an estimated noise level of 46dB would correspond to 82dB(A), resulting in the offset from 82dB downto 46dB seen in Fig. 3.7.

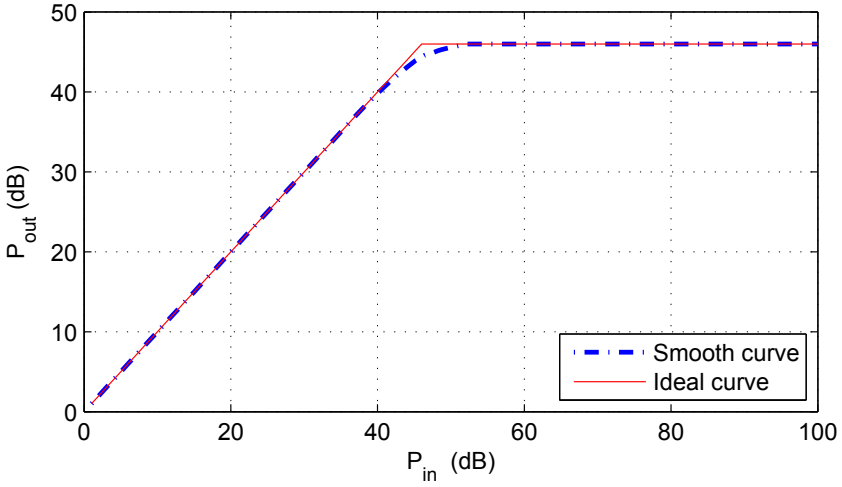


Fig. 3.7: Gain Lookup Table. P_{out} is a function of P_{in} as in (3.11).

$$P_{\text{out}} = 10 \cdot \log_{10}\left(10^{P_{\text{in}}/10}\right) \cdot LUT(P_{\text{in}}). \tag{3.11}$$

where: P_{in} = Power before gain is applied
P_{out} = Power when gain is applied
LUT = Lookup table, returns gain for corresponding P_{in}

Chapter 4

Initial Hardware Implementation

This chapter will cover the hardware implementation of the algorithm that was developed in Matlab. The Matlab code is written to be as easy as possible to translate into VHDL.

The filters and the AGC uses finite state machines (FSMs) connected together for a synchronized flow of the samples through the circuit. The blocks are synchronous sequential circuit that performs a part of the calculations in different states each clock cycle. An advantage of hardware implementation is that several calculations can be done in parallel. In total it takes 15 clock cycles for this implementation of the AGC algorithm to process one sample.

To illustrate the signal connections in a hardware block, algorithmic state machine with data path (ASMD) are used. In [5], the notation "$\leftarrow$" is used for a register transfer (RT) operation, i.e. when a signal is updated on the next clock cycle, and "$\Leftarrow$" is used for regular signal assignment, i.e. when a signal is set immediately. For an RT operation, the assigned signal is the input to a register. For example, the operation `r1` $\leftarrow$ `r2`, where one want to store the value of register `r2` in register `r1`, is implemented as `r1_next` $\Leftarrow$ `r2` and on the next clock event `r1` $\Leftarrow$ `r1_next`. In other words, `r2` is set as the input to register `r1`. At the next clock cycle, `r1` will hold the same value as `r2`.

4.1 Filters

4.1.1 High Pass Filter

The high pass filter is a first order IIR filter implemented in Direct-Form-I, see Fig. 3.3, using the FSM in Fig. 4.1 and the ASMD in Fig. 4.2. The FSM starts in the *HOLD* state and waits for the signal i_{start} to be asserted. When i_{start} goes high, the input sample is latched in and the FSM moves to the *CALC* state. In *CALC*, the current input sample, previous input sample, and previous output sample are multiplied with the corresponding filter coefficients and then added/subtracted,

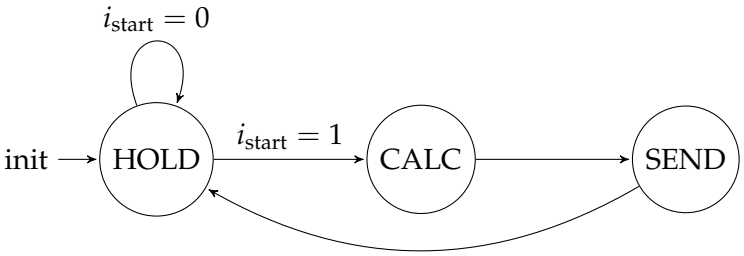


Fig. 4.1: FSM for controlling the high pass filter and the equalizer filter.

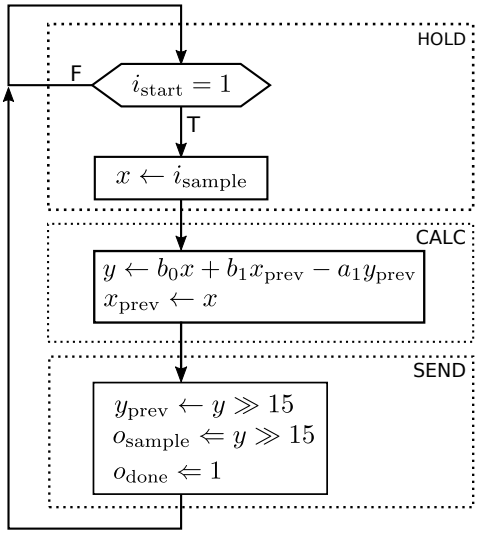


Fig. 4.2: ASMD for the high pass filter.

as in (3.2), to make the current output sample. The current input sample is also saved as the previous input sample, to be used for the next filter calculation. The state is then shifted to the *SEND* state, where the output sample is rightshifted 15 times, to compensate for the fixed point filter coefficients, and then sent to the equalizer filter. The outputted sample is also saved as the previous output sample, to be used for the next filter calculation. The o_{done} signal is asserted for one clock cycle, to let the equalizer filter's FSM know when to start, before the FSM shifts back to the *HOLD* state.

4.1.2 Equalizer Filter

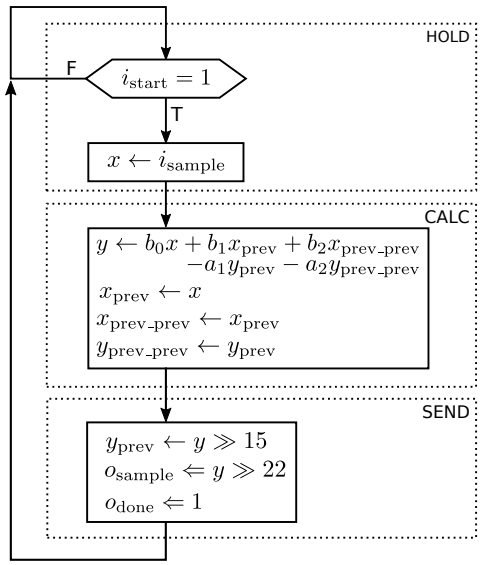


Fig. 4.3: ASMD for the equalizer filter.

The equalizer filter is a second order IIR filter implemented in Direct-Form-I, see Fig. 3.5, using the FSM in Fig. 4.1 and ASMD in Fig. 4.3. The FSM starts in the *HOLD* state and waits for the signal i_{start} to be asserted. The i_{start} signal is connected directly to the o_{done} signal of the high pass filter. When i_{start} goes high, the input sample is latched in and the FSM moves to the *CALC* state. In *CALC*, the current input sample, the two previous input samples, and the two previous output samples are multiplied with the corresponding filter coefficients and then added/subtracted, as in (3.4), to make the current output sample. The current input sample is also saved as the previous input sample, and the previous input- and output samples are saved as the before last input- and output samples, to be used for the next filter calculation. The state is then shifted to the *SEND* state, where the output sample is rightshifted 15 times, to compensate for the fixed point filter coefficients, and saved as the previous output sample, to be used for the next filter calculation. The output sample is also rightshifted an additional

7 times, to dampen the sample due to the amplification caused by the filter, before it is sent to the AGC. The signal o_{done} is asserted for one clock cycle, to let the AGC's FSM know when to start, before the FSM shifts back to the *HOLD* state.

4.2 AGC

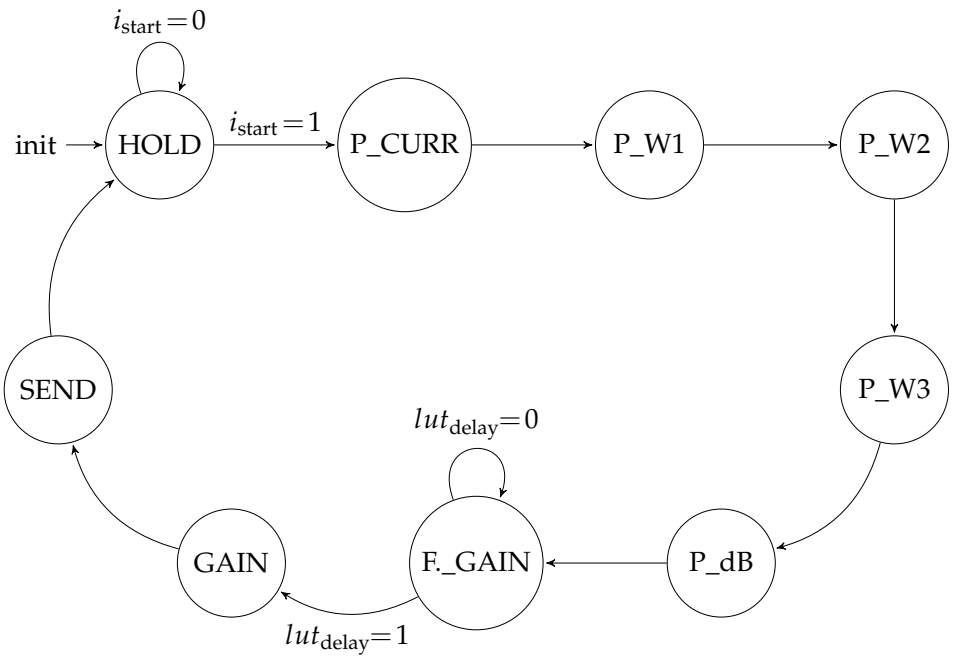


Fig. 4.4: FSM for controlling the AGC. The state name *F._GAIN* is an abbreviation for *FETCH_GAIN*.

The AGC is implemented using the FSM in Fig. 4.4 and ASMD in Fig. 4.5. The FSM starts in the *HOLD* state and waits for the signal i_{start} to be asserted. The i_{start} signal is connected directly to the o_{done} signal of the equalizer filter. When i_{start} goes high, the input sample is latched in and the FSM moves to the *P_CURR* state. In *P_CURR* the absolute value of the sample is squared, to calculate the power of that sample, P_{in}. Then the state is shifted to *P_W1*, where P_{in} is weighted with the time constant α against the weighted power of the previous sample, $P_{\text{w\_fast\_prev}}$. The FSM then shifts state to *P_W2* where the maximum value of $P_{\text{w\_fast}}$ and $P_{\text{weighted\_prev}}$ is stored as the current P_{weighted}. Note that $P_{\text{w\_fast}}$ is rightshifted 15 times to compensate for the fixed point representation of α. In the next state, *P_W3*, P_{weighted} is scaled down by the time constant $(1-\beta)$ if the sample's power is decreasing. In case of the weighting with β, $P_{\text{weighted\_next}}$ is rightshifted 15 times before assigned to P_{weighted}. When the final value for P_{weighted} is determined, the FSM shifts state to *P_dB* where P_{weighted} is converted to decibel

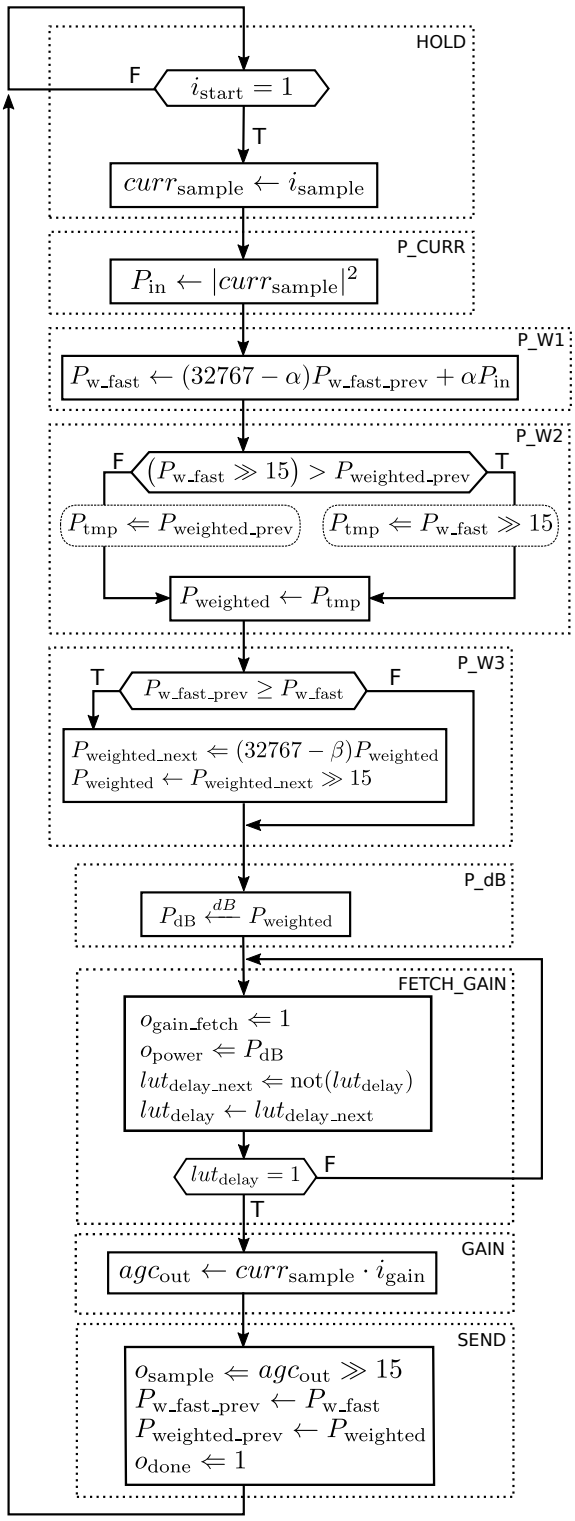


Fig. 4.5: ASMD for the AGC.

scale. After that, the state is shifted to *FETCH_GAIN*, where the decibel value is sent to the Gain LUT and an enable signal for the LUT is sent. The LUT has one clock cycle read delay, therefore a one-bit counter, lut_{delay}, is used to stall the FSM for one clock cycle before shifting to state *GAIN*. In *GAIN*, the fetched gain from the LUT is multiplied with the input sample before shifting state to *SEND*. In *SEND* the product from the multiplication is latched out after being rightshifted 15 times, to compensate for the applied gain's fixed point representation. $P_{\text{w\_fast}}$ and P_{weighted} are store in $P_{\text{w\_fast\_prev}}$ and $P_{\text{weighted\_prev}}$ respectively, to be available when the next sample is processed. After that the FSM shifts back to the *HOLD* state.

There is no hardware friendly method of calculating logarithms, which is needed when converting the sample's power into decibel scale, as in (3.8). An easy workaround for this is to use a number of comparators, starting with checking if the power of the sample is at its highest possible value. If it is that high, then the corresponding decibel-value is assigned to the target register. Otherwise, the power will be compared to the next corresponding decibel value, and so on. The maximum value for the power of a sample is $2^{32-1} - 1 = 2\,147\,483\,647$, since it is stored in a 32-bit signed number. This corresponds to $10 \cdot \log_{10}(2^{32-1} - 1) \approx$ 93.32dB. To include round-off, a sample-power that is larger then the corresponding value of 92.5dB (1 778 279 410) will be the first value to be compared to, resulting in 93dB if larger. If it is smaller then 92.5dB but larger then 91.5, the power of the sample is rounded to 92dB. See the pseudo code in Listing 4.1. There is no point of comparing all the way down to 0dB since at a certain power level, depending on the LUT, the gain for this level and any below this should be 1.

Listing 4.1: Pseudo code for decibel conversion.

```
if P_sample > 1778279410 then        // >92.5dB
    P_sample_dB <= 93;
else if P_sample > 1412537545 then   // >91.5dB
    P_sample_dB <= 92;
    .
    .
    .
else if P_sample > 14125 then        // >41.5dB
    P_sample_dB <= 42;
    .
    .
    .
else if P_sample > 2 then            // >3dB
    P_sample_dB <= 3;
else
    P_sample_dB <= 0;
```

4.3 Gain Lookup Table

The lookup table (LUT) is a read only memory (ROM) with a gain corresponding to the power of a sample measured in dB. The functionality of the LUT is shown in Fig. 4.6. It is divided into two parts running in parallel, one for finding the largest decibel level, and one doing the lookup using the largest decibel level. The LUT has two enable signals and two decibel level inputs, one from each stereo channel (left and right). When any of the two enable signals are asserted, both the decibel level inputs are compared to determine which is the largest. The channel with the largest power will determine the gain for both channels. This equal gain factor for both left and right channel will create a natural difference in noise level for both ears, when still keeping the ability for the user to locate the direction of the noise source. The maximum power is stored in a register which is used in the lookup process. The corresponding gain for the maximum power is returned to both the left and right channel simultaneously.

The tables are generated as floating point numbers in Matlab, as described in section 3.4, and converted into 16-bit fixed point numbers.

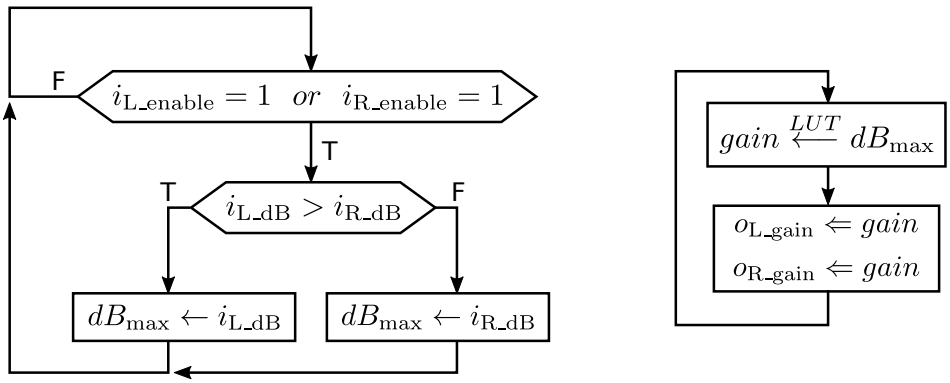


Fig. 4.6: Algorithmic flowchart of signals in the gain lookup table. To the left is the updating the maximum noise level and to the right is the looking up and sending the corresponding gain.

4.4 Hardware Resources

To find out how many hardware components that are required to realize the initial AGC implementation in hardware, the behavioral model was run through the *Synopsis Design Compiler* synthesis tool. The resources used by the synthesis tool, with an estimated chip area, is listed in Table 4.1. By studying the behavioral model, it was expected that a total of 1081 registers, 10 adders, 6 subtracters, 26 multipliers, and 117 comparators would be utilized. The difference in register count is because some signals are trimmed down by the synthesis tool since

not all their bits are used. The full length is however needed in the behavioral model to have the correct size of the output from a multiplier for instance. In the synthesized netlist, these registers are not needed since they do not connect to anything.

TABLE 4.1: Hardware resources used in initial design. The *Stereo AGC* consists of two identical designs. By dividing these resources with 2, the resources for a single AGC implementation is given.

	Number of resources used		
Resource	Stereo AGC	Gain LUT	**Total**
Register	930	23	**953**
Adder	10	—	**10**
Subtracter	6	—	**6**
Multiplier	26	—	**26**
Comparator	116	1	**117**
Estimated area: 0.0578mm^2			

Chapter 5

Optimizing Hardware Implementation

This chapter covers the optimizations to the design that are made to make the implementation of the algorithm more efficient in terms of circuit area and power dissipation.

5.1 Resource Sharing

The initial implementation needs several adders and multipliers, see Table 4.1, and multipliers in particular can be very large in terms of area. The idea of resource sharing is that instead of having many parallel adders and multipliers, by using time-multiplexing, one adder and one multiplier can be used repeatedly in separate states to perform the same calculations [5]. For example, the high pass filter in section 4.1.1 uses three multipliers and two adders in parallel to filter a sample. By using one multiplier and schedule the multiplications in three different time-slots (states) and accumulating the products using one adder, less resources are needed. To achieve this resource sharing, some additional registers (to store intermediate values) and multiplexers (MUXs) (to route inputs and outputs to/from the arithmetic units) are needed. However, multipliers used for multiplying an input with a constant, such as filter coefficients, are optimized by the synthesis tool and are therefore not as large as a general multiplier with two inputs. For example, a multiplication with 2^n, $n \in \mathbb{Z}^+$ is optimized as a leftshift n times. To achieve a multiplication not equal to a power of two, shifting can be combined with addition or subtraction, like:

$$x \cdot 5 = x \ll 2 + x = 4x + x = 5x. \tag{5.1}$$

In fact, 22 out of the 26 multipliers in Table 4.1 are constant multipliers and replacing these with one general multiplier will probably not reduce the total area. The remaining four general multipliers can on the other hand be beneficial to combine into one, since they are quite large (16×16 bits each). Since resource sharing will be implemented for the general multipliers, the constant multipliers might as well be included.

The number of adders and multipliers will decrease but the number of registers will increase. There is also a less parallel execution of the algorithm and therefore the execution-time will increase. Instead of 15 clock cycles, this resource sharing algorithm takes 38 clock cycles to process one sample. In order to minimize the usage of input- and output (I/O) pads for the ASIC, the samples will also be read and outputted in serial rather than parallel. More on the I/O pads in section 7.1.2. This serial read and write adds an additional 32 clock cycles to the algorithm for a total of 70 clock cycles. As always there is a trade-off between performance and size.

5.1.1 Resource Sharing Algorithm

With only one adder and one multiplier per audio channel plus the serial read and write of samples, the FSM has to be redesigned. To share as much resources as possible, the filters and AGC are joined together in one block instead of dividing the algorithm into different blocks. The overall behavioral is still the same, it just takes longer time since fewer arithmetic units are available, thus less parallel calculations are done.

When only having one adder and one multiplier to do all calculations, they have to be large enough to accommodate for all possible inputs. By tracing all signal paths and opting for a worst case scenario, i.e. maximum amplitude of audio samples, it is found that one input to the multiplier has to be 32 bits and the other input 23 bits. This is the largest two inputs to the multiplier that are routed at the same time. However, these 23 bits are needed for representing the large filter coefficients for the equalizer filter. Without this filter, 16 bits would be enough to represent the remaining signals. Simulations in Matlab shows no noticeable difference when reducing the size of the filter coefficients by scaling them with a factor 2^8 instead of 2^{15} during the conversion to fixed point numbers. This gives the following filter coefficients for the equalizer filter in (3.4):

$$\begin{aligned} a_1' &= -156 \\ a_2' &= -75 \\ b_0' &= 27742 \\ b_1' &= -156 \\ b_2' &= -27561. \end{aligned}$$

These coefficients can be represented using 16 bits, resulting in a 32×16-bit multiplier with a 48-bit output. The high pass filter's coefficient can already be represented using 16 bits and therefore no altering of them are needed. Using the same methodology, following all signal paths, the largest two inputs to an adder at the same time are two 48-bit numbers. This occurs when summing the outputs from the adder and multiplier together. If a signal for multiplication or addition is smaller (fewer bits) then the corresponding input it is routed to, the signal is sign-extended, i.e. padded with its MSB, to the correct size.

The inputs for the adder and multiplier are registers called $add_{\text{src1,2}}$ and $mult_{\text{src1,2}}$, and the outputs are registers called add_{out} and $mult_{\text{out}}$ respectively in the ASMD in Fig. 5.2 and 5.3. They are routed using multiplexers that are controlled depending on which state the FSM, Fig. 5.1, is in. If the product from the the multiplier is used in the next state, there is a one clock cycle delay inserted to allow the $mult_{\text{out}}$ register to be updated.

As in the previous design, the FSM starts in the *HOLD* state, waiting for i_{start} to be asserted before moving to state *LATCH_IN_SAMPLE* where the sample is read in serial, starting with the MSB. The 4-bit counter $inout_{\text{cnt}}$ is used to index each bit to its correct register. When the counter reaches 15, all bits of the sample are read and $inout_{\text{cnt}}$ should stay at 15. After that the state is shifted to *HP_CALC1*, where the first multiplication with one of the filter coefficients are made. The output from the multiplier is one of the inputs for the adder in the next state, therefore there is a one clock cycle delay before going to the *HP_CALC2* state. In *HP_CALC2*, $mult_{\text{out}}$ is assigned as one of the inputs for the adder. The other input is zero, since this is the first accumulation of multiplications for the filter. The multiplier also do the next multiplication for the filter, then the state shifts to *HP_CALC3* where the output from the adder and multiplier are summed. This pattern repeats until both of the filters are done. In *EQ_CALC1* and *FINISH_CALC*, add_{out} is rightshifted to compensate for the fixed point filter coefficients, and also saved as the previous samples for the next time a sample is processed. In *FINISH_CALC* the previous samples are also saved as the before last samples, and the current output sample is rightshifted 7 additional times to dampen most of the amplification caused by the equalizer filter.

With the filters done, the FSM shifts to the *P_CURR* state where the absolute value of the current sample is squared, to calculate the power of the sample, before moving to state *P_W1*. The weighted power calculations are performed in the same way as the filters, multiplying and accumulating, and finishes in the *P_W4* state where add_{out} is rightshifted 15 times, to compensate for the fixed point time constant α, and assigned to $P_{\text{w\_fast}}$. In case P_{weighted} is to be weighted with β in a later state, the multiplication is started here to avoid delays and since the multiplier is free. The decision box comparing $\left(add_{\text{out}} \gg 15\right)$ and $P_{\text{weighted\_prev}}$ determines the next state and which signal to be routed to $mult_{\text{src1}}$. Either *P_W_INCR* with $mult_{\text{src1}}$ set to $add_{\text{out}} \gg 15$ if $add_{\text{out}} \gg 15$ is the largest, or *P_W_DCR1* with $mult_{\text{src1}}$ set to $P_{\text{weighted\_prev}}$ if $P_{\text{weighted\_prev}}$ is the largest. In both *P_W_INCR* and *P_W_DCR1*, P_{weighted} is assigned the largest value in the previous decision box. The next decision box, comparing $P_{\text{w\_fast\_prev}}$ and $P_{\text{w\_fast}}$, determines if P_{weighted} should be weighted against β or not. If so, the next state is *P_W_DCR2*, otherwise *P_dB*. Regardless of which of the two state the FSM is in, the multiplication for the weighting with β is already performed. If the next state is *P_W_DCR2*, P_{weighted} is assigned to $mult_{\text{out}}$ after 15 rightshifts, to compensate for the fixed point time constant β. After that, the FSM shifts state to *P_dB*. Here the final weighted power, P_{weighted}, is converted to decibel scale using comparators. The decibel value is stored in a register called P_{dB}. The state is then shifted to *FETCH_GAIN* where P_{dB} and the enable signal ($o_{\text{gain\_fetch}}$) is sent to the gain LUT. There is a one clock cycle delay here, to wait for the LUT to return the gain, before the

FSM moves to the *GAIN* state. In *GAIN*, the fetched gain from the LUT is multiplied with the current sample. The FSM is stalled one clock cycle before shifting to *LATCH_OUT_SAMPLE*, since the product is needed in that state. When in *LATCH_OUT_SAMPLE*, the output from the multiplier is rightshifted 15 times, to compensate for the fixed point gain, and then outputted in serial, starting with the MSB. In this case, $inout_{\text{cnt}}$ starts at 15 and decreases in order to avoid glitches on the output pin when indexing $mult_{\text{out}}$ for latching out the sample. A subtraction in the index (like in *LATCH_IN_SAMPLE*) would introducing a short delay, causing the output signal to be pulled low for a very short time between sending high signals. These glitches was found during post-layout-simulations and was after that fixed by reversing $inout_{\text{cnt}}$ in the algorithm. At the first iteration, $P_{\text{weighted\_prev}}$ and $P_{\text{w\_fast\_prev}}$ are set to P_{weighted} and $P_{\text{w\_fast}}$ respectively. At the same time the LSB of the output sample is sent, the o_{done} signal is asserted, $inout_{\text{cnt\_next}}$ is set to zero, and the next state is set to *HOLD* again.

5.1.2 Results of Resource Sharing

The major hardware components used with the resource sharing implementation, and the difference compared to the initial implementation, is listed in Table 5.1. To make a good comparison, the same timing constrains as in the initial design are used. The area with resource sharing is smaller, but were at first expected to be a lot smaller, due to the number of multipliers that were removed. However as previously discussed, most of the multipliers, 22 out of 26, are constant multipliers. For example, in the filters the samples are multiplied with constant filter coefficients and in the AGC the powers are weighted against fixed time constants. These multipliers are optimized to calculate one input against a constant value, thus no need for a complete multiplier that takes up a lot more area.

Studying the behavioral model, it was expected that the synthesis tool would utilize 1275 registers, 2 adders, 2 subtracters, 2 multipliers, and 119 comparators. The only difference between the resources used by the synthesis tool and what was expected is two comparators. This is probably the result of one comparator in each AGC being used twcie. When deciding the next state in both state *P_INCR* and *P_DCR1*, see Fig. 5.1, the same comparison is done, meaning there is no need to have two identical comparators in two different states.

TABLE 5.1: Hardware resources used in resource sharing design. The *Stereo AGC* consists of two identical designs. By dividing these resources with 2, the resources for a single AGC implementation is given. *Diff. from Table 4.1* shows the differences when resources sharing is implemented.

	Number of resources used			Diff. from
Resource	Stereo AGC	Gain LUT	**Total**	Table 4.1
Register	1252	23	**1275**	+322
Adder	2	—	**2**	-8
Subtracter	2	—	**2**	-4
Multiplier	2	—	**2**	-24
Comparator	116	1	**117**	0
Estimated area: 0.0414mm^2				-28%

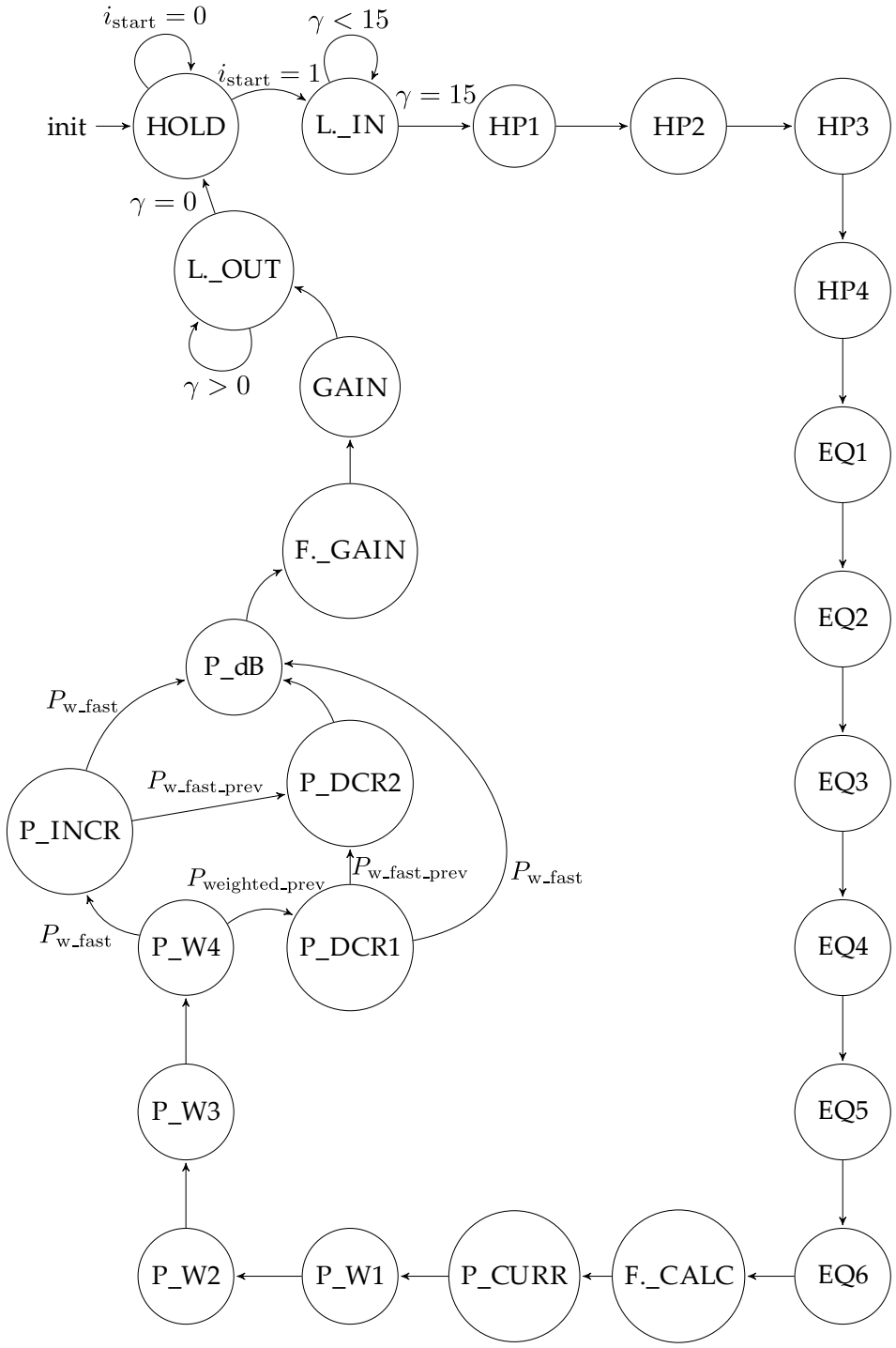


Fig. 5.1: FSM for controlling the resource sharing AGC. γ is a four-bit counter (0–15) for reading and writing serial data. $P_{\text{w...}}$ denotes the largest power when branching in state *P_W4*, *P_DCR1*, and *P_INCR*. See Table 5.2 for a short description of each state.

TABLE 5.2: Description of AGC's FSM states from Fig. 5.1.

State	State name	Description
HOLD	Hold	Waiting for start signal
L._IN	Latch_in_sample	Read input sample, 16 bits serial data
*HP*1	HP_calc1	Calculations for the high pass filter
*HP*2	HP_calc2	
*HP*3	HP_calc3	
*HP*4	HP_calc4	
*EQ*1	EQ_calc1	Calculations for the equalizer filter
*EQ*2	EQ_calc2	
*EQ*3	EQ_calc3	
*EQ*4	EQ_calc4	
*EQ*5	EQ_calc5	
*EQ*6	EQ_calc6	
F._CALC	Finish_calc	Save current sample as prev. sample
P_CURR	P_current	Calculate power of current sample
*P_W*1	P_w1	Weighting, fast increasing power
*P_W*2	P_w2	
*P_W*3	P_w3	
*P_W*4	P_w4	
P_INCR	P_increase	Weighting, increasing power
*P_DCR*1	P_decrease1	Weighting, decreasing power
*P_DCR*2	P_decrease2	
P_dB	P_dB	Transform weighted power to decibel
F._GAIN	Fetch_gain	Fetch gain from LUT
GAIN	Gain	Multiply current sample with gain
L._OUT	Latch_out_sample	Write output sample, 16 bits serial data

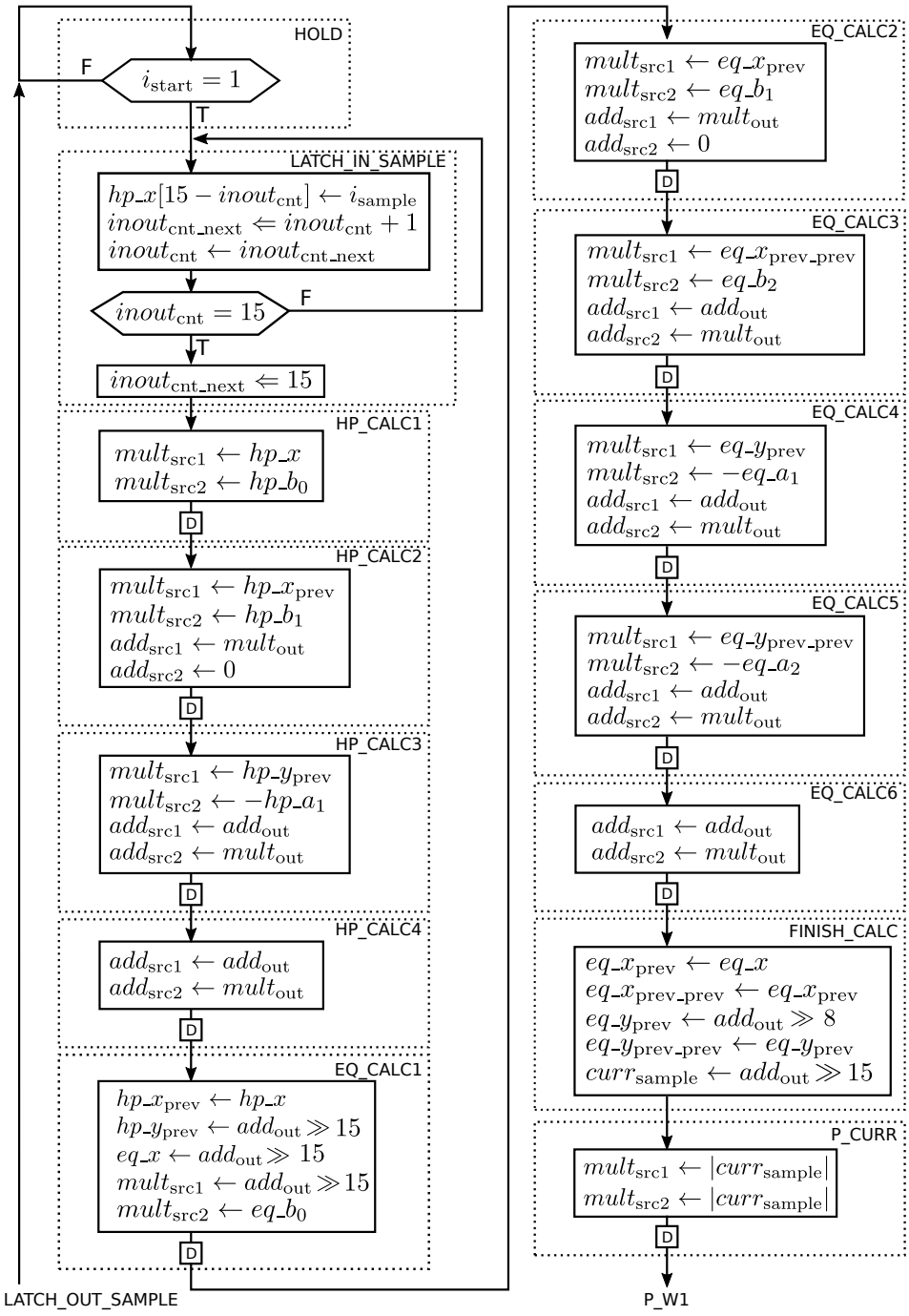


Fig. 5.2: ASMD for the AGC with resource sharing. Part 1 of 2. D denotes a delay of one clock cycle.

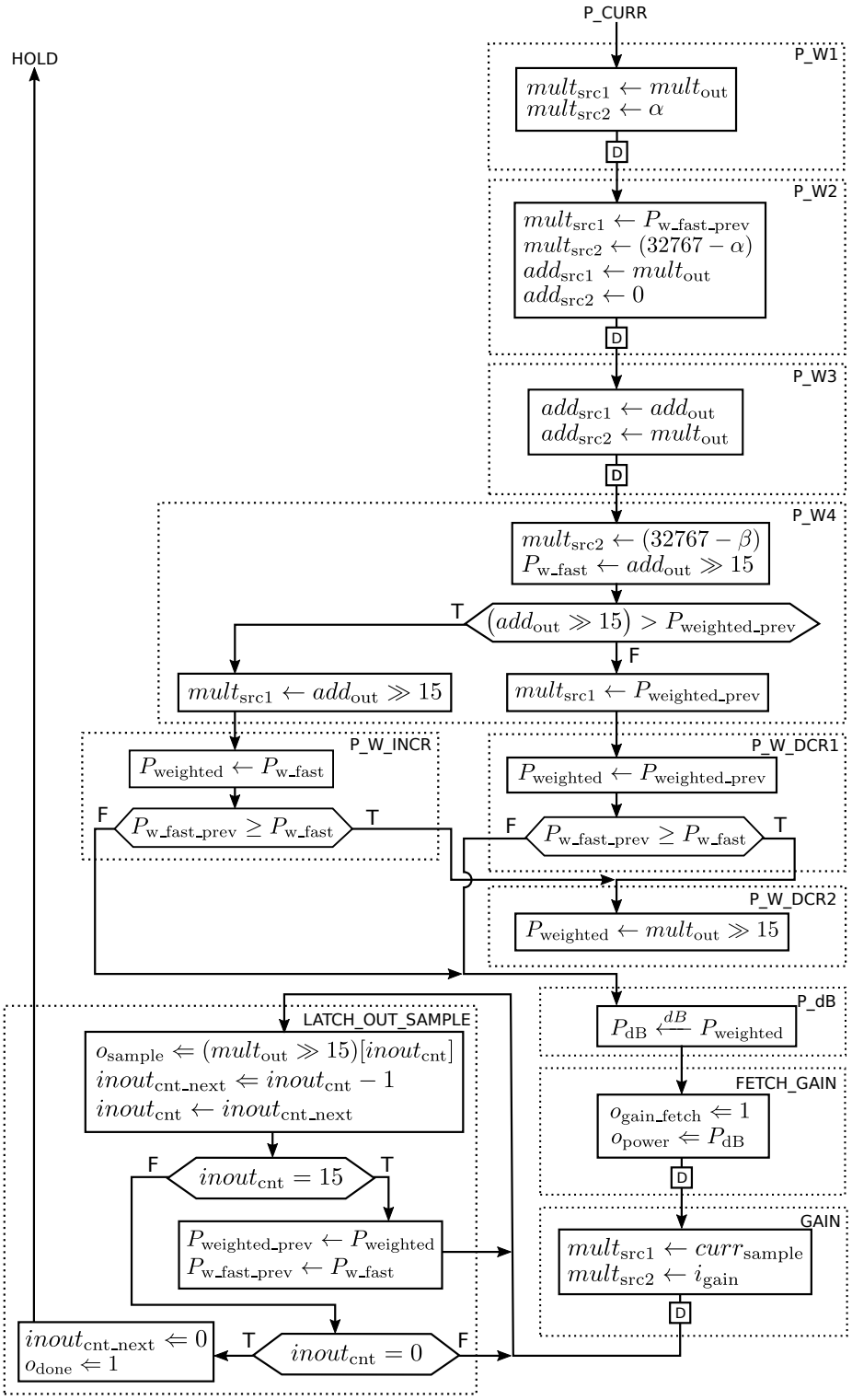


Fig. 5.3: ASMD for the AGC with resource sharing. Part 2 of 2. D denotes a delay of one clock cycle.

5.2 Scaling Supply Voltage

The power consumption of an integrated circuit consist of the dynamic power and static power, see (5.2). Dynamic power is a result of charging or discharging the gate-capacitance when the CMOS switches from $0 \rightarrow 1$ and $1 \rightarrow 0$, along with the short-circuit current during switching. The static power comes from leakage current when the CMOS is not switching. [6]

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} \tag{5.2}$$

where the dynamic power is relative to the clock frequency and supply voltage:

$$P_{\text{dynamic}} \propto f_{\text{clk}} \cdot V_{\text{DD}}^2 \tag{5.3}$$

and the static power is the result of the leakage current when the transistors are not switching, i.e.,

$$P_{\text{static}} = I_{\text{leakage}} \cdot V_{\text{DD}}. \tag{5.4}$$

As seen in (5.3), the supply voltage has a quadratic effect on the dynamic power. By lowering V_{DD} by half, the dynamic power will go down to one-forth. This is with the assumption that the clock frequency stays the same. Usually, a slower clock is needed since the transistors operates slower when powered with a lower voltage. With a slower clock, the power savings are even bigger due to the proportional relation between dynamic power and switching frequency.

To not lose too much performance when scaling down the supply voltage, the gate threshold voltage (V_{T}) can be lowered in order for the transistor to operate quicker. However, a lower threshold voltage introduces more leakage current. Three different sets of libraries, with different V_{T}, will be used when scaling the supply voltage in order to decrease the power consumption:

LPHVT Low power cell library with high threshold voltage transistors. Has a slow operating speed and low leakage.

LPSVT Low power cell library with standard threshold voltage transistors. Has a medium operating speed and medium leakage. SVT is about ten times faster then HVT, with approximately ten times the leakage.

LPLVT Low power cell library with low threshold voltage transistors. Has a fast operating speed and high leakage. LVT is about ten times faster then SVT, with approximately ten times the leakage.

It is desirable to run the circuit with the fastest clock frequency possible to minimize the leakage current. The clock frequency is limited by the critical path in the design. Some components are faster then others, for example adders are faster then multipliers, but with a well balanced design, as many as possible of the transistors in the design will be done switching at the same time. This results in a proportionally lower leakage power compared to the dynamic power since fewer

transistors are not switching. With a clock frequency that is too slow, all components in the design will be done with time to spare before the next transition of the clock, resulting in a circuit that leaks unnecessary current.

The problem when scaling the supply voltage is that at a certain level, the transistors will switch so slowly that the leakage power increases more than the dynamic power decreases. It is therefore desirable to run the circuit at the voltage level of this sweet spot, if the timing constrains are met of course. Using the aforementioned cell libraries, several supply voltages will be used and compared in order to find the most power efficient implementation. It might be possible that switching to a faster library with more leakage could lower the total power dissipation thanks to the ability to perform at lower voltages.

Chapter 6

Hardware Implementation on FPGA

In addition to the filters and AGC algorithm, an interface for the FPGA's audio codec is also needed to run the design on the FPGA. This chapter will cover the controller and the other additional components necessary for the algorithm to be fully functional, and to be able to connect the hearing protectors.

6.1 FPGA Board

For testing and verification of the hardware, a Xilinx XUPV5-LX110T Evaluation Platform from Digilent with a Virtex 5 FPGA from Xilinx is used, see Fig. 6.1. It has an on-board AC'97 codec, *AD1981B* from Analog Devices, with peripheral jacks for microphone input, line in, line out, headphone, and SPDIF digital audio output, see Fig. 6.2. The only jacks to be used in this thesis are the microphone and headphone jacks, both with a stereo 3.5mm female connectors. The microphone jack has a 2.25V biasing voltage[1] , and the headphone jack is driven by a 50mW amplifier. The AD1981B chip contains a 16-bit ADC and a 20-bit DAC. [7] [8]

Apart from hardware simulations, all the changes during development of the hardware design needs to be tested on physical hardware to verify its functionality. This is important since it is difficult to write simulation-testbenches that covers every aspect of the design. If the design do not work on an FPGA, it is not going to work on an ASIC.

[1] Can also be configured with 3.7V biasing voltage.

Fig. 6.1: Xilinx XUPV5-LX110T Evaluation Platform with Xilinx Virtex 5 XC5VLX110T FPGA. [9]

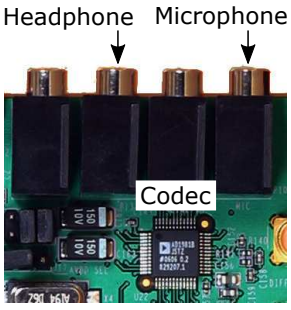


Fig. 6.2: Codec, headphone jack and microphone jack marked out on the FPGA.

6.2 AC'97 Controller

The FPGA used for verification has an AC'97 codec, an AD1981B chip, for audio processing. This is a codec developed by Intel and was previously the standard audio codec to be used on for instance a computer's motherboard. The AD1981B chip has a 16-bit ADC and a 20-bit DAC[2], both with a sample rate of up to 48kHz.

In order to interface with the AC'97 codec, a controller is needed. A design for a controller to interface the *LM4550 AC'97 codec* on an *ATLYS Spartan 6 FPGA* was found at [10]. This is another AC'97 codec but the link interface is the same, so it will work as the backbone of the implementation needed in this thesis. The link between the codec and the controller uses a fixed bitrate of 12.288Mbit/s divided into 256-bit frames, giving it a fixed frame frequency of $\frac{12.288\text{Mbit/s}}{256\text{bit}} = 48\text{kHz}$. The link interface (marked with $\boxed{LINK}$ in Fig. 6.4) consists of five wires:

bit_clock, 12.288MHz clock signal from the codec to the controller.

sync, signal from controller to codec to synchronize when a new frame begins.

reset, signal from controller to codec for resetting the codec.

sdata_in, serial data from codec to controller.

sdata_out, serial data from controller to codec.

Each 256-bit frame is divided into 13 slots containing different data, first a 16-bit slot (slot 0) and then twelve 20-bit slots (slot 1 – slot 12). The frames are transmitted on *sdata_in* and *sdata_out* simultaneously using pulse-code modulation (PCM). Slot 0 specifies which slots, if any, that contains valid data. Slots 1 and 2 are used for writing to the codecs control registers, or reading status of control registers. See section 6.2.3 for details on the registers. Slot 3 is used for sending PCM data to the left channel DAC or receiving PCM data from the left channel ADC. Slot 4 has the same properties as slot 3 but for the right channels data instead. The remaining slots are used for surround sound channels and not used in this implementation.

6.2.1 Inverting Bit Clock

The convention when sending and receiving serial data on the links between the codec and the controller is to send on the rising edge of the bit clock, and to read data on the falling edge. This applies for both the codec and the controller. This means that when receiving data from the DAC, the process responsible for latching in data has to trigger on the falling edge of the bit clock in order for the data to be read accurately.

It is not a good idea to mix processes that trigger on rising edge and falling edge in the same design. This is solved by having a block that inverts the bit clock

[2]Only 16-bit resolution is used for the DAC since the ADC has a maximum of 16 bits.

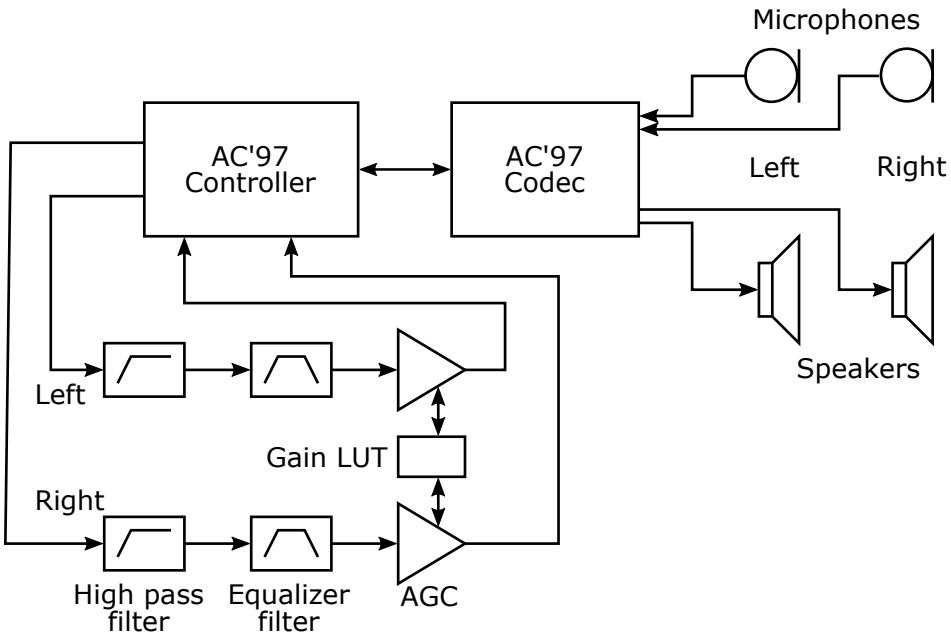


Fig. 6.3: Overview of AC'97 codec and controller connected to filters and AGC.

signal, so the controller has both the regular bit clock signal and the inverted bit clock signal as inputs. Thereby, the processes can use whichever clock signal they need, all of them triggering on rising edges.

6.2.2 Connecting Peripherals

Numbers and letters marked as [X] refers to the squares in Fig. 6.4. By default, all paths with a mute-switch are muted except for the mute-switches [6] connecting the DAC outputs to the mixers [Σ].

Microphone and Headphone

The microphone jack [IN] is by default selected as the input for the ADC by the MUX [2]. However, it is necessary to unmute the input to the ADC [3] and to enable stereo microphone input by selecting the MIC2 input on the MUX [1].

When a sample returns from the DAC, the mute-switches [4] has to be unmuted, and the output selector MUX [7] routes the DAC to the output jacks. To play the sound through the headphones [OUT], the mute-switches [8] are unmuted.

Volume

During development and testing it is desirable to be able to change the volume of the output sound. The codec has a register that controls the attenuation [8] for the headphone volume, with five bits for left channel volume and five bits for the right channel volume. Using five switches on the FPGA, connected to both channels volume bits, the output volume to the headphones can be controlled.

AGC on/off

To evaluate the function of the AGC there should be an easy way of turning the AGC on or off. One switch on the FPGA is used for bypassing the AGC, or to be more precise, bypassing the ADC and DAC. This switch inverts the state of the mute-switches [3], [4], and [5], turning off all inputs to the ADC and outputs from the DAC, and activates the microphone input to the mixers [Σ]. The mute-switches [6] are activated, since they are disabled by default. The output MUX [7] also routes the mixers to the output. The microphone is then connected directly to the outputs. Note that only one of the microphones, [IN], can be used as input but the sound is played back through both left and right channel on the headphones.

Reset

In case of an error, the user should be able to reset the AC'97 codec and AGC algorithm. One switch on the FPGA is therefore connected to the design's reset input.

6.2.3 AD1981B Control Registers

The AD1981B codec has twenty-seven 16-bit control registers for customizing its functionality. After a reset, all registers return to their default state, meaning no sound goes through the codec until it is configured correctly. The FSM in Fig. 6.5 is used to cycle through the registers that needs to be updated in order for the codec to work with the AGC algorithm. Each state contains the data that should be written to a register, along with the address to that register. Information on what bits that are to be set in the registers, and their functions, are shown in Table 6.1 – 6.7. Some functions, like headphone volume and turning the AGC on or off, should be controllable without resetting the codec in between. This is solved by having some of the states in the FSM reading the switches on the FPGA and updating the data before it is written to the codec's registers.

The link between the AC'97 controller and codec allows for changing the data of one register each frame, during slots 1 and 2. When a new frame is about to begin, the FSM switches state upon request by the AC'97 controller, and sends

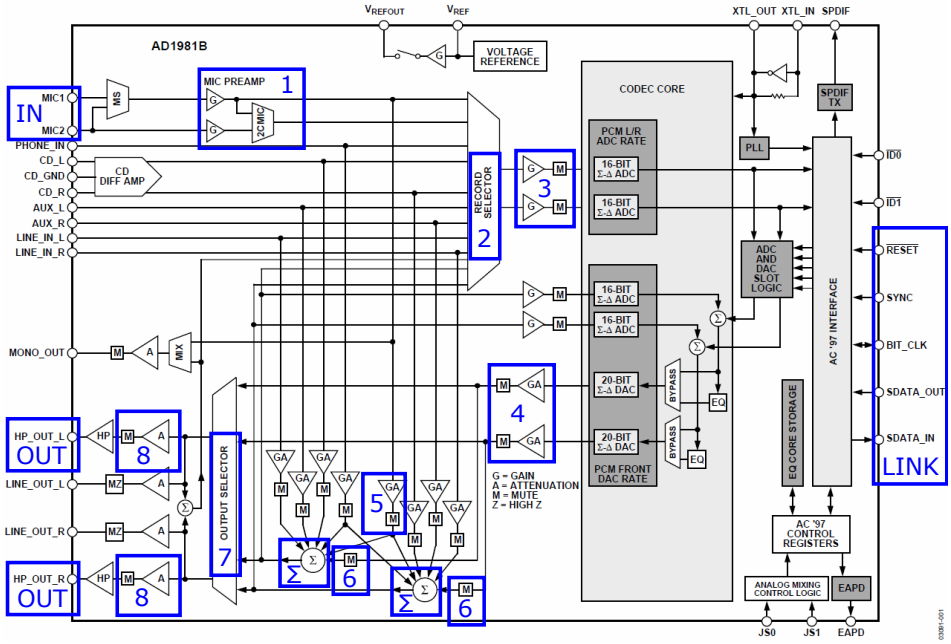


Fig. 6.4: Schematic over AD1981B codec. [8]

the register address and the register data to the AC'97 controller. The AC'97 controller then sends the data to the correct register.

Only the bits in the registers that are used are represented in the tables. There are more control bits in the registers that are not marked out in the tables. Remaining registers use their default values. Consult [8] for all registers and more details.

TABLE 6.1: Headphone volume register, address 0x04
M = mute headphone output 8
Use settings "0x0000" for maximum volume.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
M	–		Left volume					–			Right volume					0x8000

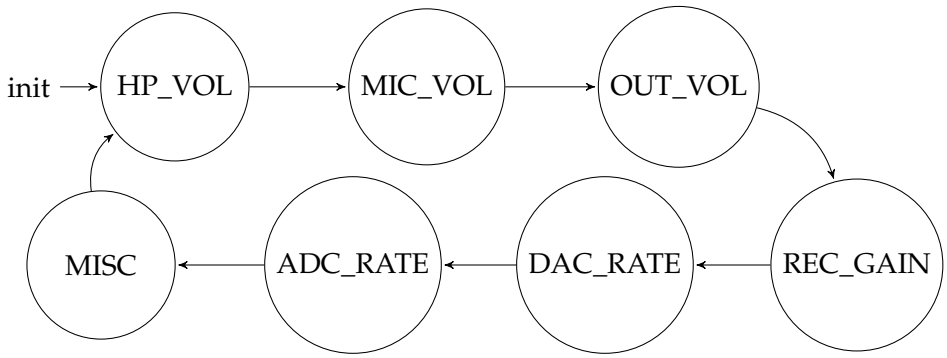


Fig. 6.5: FSM for setting data in AC'97 codec registers. The state is shifted when a new frame is about to begin, at the request the AC'97 controller. Information about the registers are shown in Table 6.1 – 6.7.

TABLE 6.2: Microphone volume register, address 0x0E
M = mute microphone [5] to mixers [Σ]
G = microphone preamplification [1]
Use settings "0x8000" to mute to mixers and 0dB gain.
Use settings "0x0000" to connect mic to mixers and 0dB gain.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
M	–								G	–	Mic volume					0x8008

TABLE 6.3: PCM-out volume register, address 0x18
M = mute DAC output [4]
Use settings "0x0808" to unmute and 0dB gain.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
M	–		Left volume					–			Right volume					0x8808

TABLE 6.4: Record gain register, address 0x1C
M = mute ADC input [3]
Use settings "0x0000" to unmute and 0dB gain.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
M	–			Left gain				–				Right gain				0x8000

The sample frequency, measured in hertz, for the DAC and ADC in Table 6.5 and 6.6 is represented with a 16-bit binary number. The range is between $1B58_{16} = 7000_{10} \rightarrow BB80_{16} = 48000_{10}$, with 1Hz resolution.

TABLE 6.5: PCM front DAC rate register, address 0x2C.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
Sample rate																0xBB80

TABLE 6.6: PCM ADC rate register, address 0x32.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
Sample rate																0xBB80

TABLE 6.7: Miscellaneous control bit register, address 0x76
DO = DAC as output $\boxed{7}$
DX = mute DAC $\boxed{6}$ to mixer $\boxed{\Sigma}$
2M = dual microphone input $\boxed{1}$
Use settings "0x0A40" for DAC output and dual microphones.
Use settings "0x0240" for mixer output and dual microphones.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Default
–				DO	–	DX	–		2M	–						0x0000

Chapter 7

Hardware Implementation on ASIC

This chapter will cover the workflow of implementing the optimized AGC algorithm on an ASIC.

7.1 Additional Configurations

For the ASIC to be as general as possible, the AC'97 controller is not needed and is therefore not included. Otherwise the ASIC would only work with the exact same codec as on the FPGA board, and its usefulness would be limited.

Other changes to the design consists of choosing the optimal clock frequency and adding input- and output (I/O) pads.

7.1.1 Clock Speed

The algorithm takes 70 clock cycles to process one sample, from when it starts reading the input sample until the output sample is completely sent. The equalizer filter has a bandwidth of approximately 4kHz, which means that the sampling frequency has to be at least twice that, 8kHz, according to the Sampling Theorem [11]. This sample rate or higher has to be met when determining the ASIC's clock frequency. First the time between two samples has to be determined:

$$f_s = \frac{1}{t_\mathrm{s}} \geq 8000\mathrm{Hz} \Rightarrow t_s \leq \frac{1}{8000\mathrm{s}^{-1}} = 125\mu\mathrm{s}. \tag{7.1}$$

where: f_s = sample frequency of ADC/DAC
t_s = sample period of ADC/DAC

$$t_{\text{sample}} = n_{\text{cycles}} \cdot t_{\text{clock}} = 70 \cdot t_{\text{clock}}. \tag{7.2}$$

where: t_{sample} = time to process one sample
n_{cycles} = number of clock cycles
t_{clock} = clock period for ASIC

By assuming that t_{s} from (7.1) is equal to t_{sample} from (7.2) and by combining (7.1) and (7.2), the longest clock period, i.e. slowest clock frequency, is given as:

$$t_{\text{clock}} \leq \frac{t_{\text{sample}}}{n_{\text{cycles}}} = \frac{t_{\text{s}}}{n_{\text{cycles}}} = \frac{125\mu\text{s}}{70} \approx 1786\text{ns} \Rightarrow f_{\text{clock}} \geq \frac{1}{t_{\text{clock}}} = 560\text{kHz}. \tag{7.3}$$

This means that the clock frequency for the ASIC has to be 560kHz in order to match an ADC/DAC sample rate of 8000 samples/s. Using a faster clock allows for a higher sample rate of the ADC and DAC, hopefully resulting in better sound quality. As an example, the upper limit for sample frequency of the ADC and DAC in the AD1981B codec is 48kHz. Using (7.3) with 48kHz sample rate yields:

$$t_{\text{clock}} \leq \frac{1/48000\text{s}^{-1}}{70} = \frac{20.833\ldots\mu\text{s}}{70} \approx 298\text{ns} \Rightarrow f_{\text{clock}} \geq \frac{1}{t_{\text{clock}}} = 3.36\text{MHz}. \tag{7.4}$$

7.1.2 I/O Pads

Input- and output pads takes up a lot of area, and also consumes some energy. The number of pads should therefore be kept to a minimum. One pad for supply voltage (VDD) and ground (GND) respectively is necessary, as well as one pad for the clock (CLK) and one for reset (RSTN). To save on the number of pads, the samples are read and outputted on serial pads, one for input and one for output, rather than 16 parallel pads each. There is also a *start* pad and a *done* pad, to let the algorithm know when to start latching in the input sample and to notify when the last serial output bit of the sample is sent. As the implementation supports stereo sound, these last four pads are duplicated, one set for the left audio channel and one for the right, giving that a total of 12 pads is needed for the ASIC.

7.2 Synthesis

Synthesis was performed with both standard- and re-characterized cell libraries using Synopsis Design Compiler. The compiled code results in a netlist with the same functionality as the behavioral model, but build out of logic gates. The

re-characterized cell libraries are used when scaling the supply voltage in order to get lower power dissipation. With a lower supply voltage, large logic gates with high fan-in and/or high fan-out may fail to operate due to the voltage drop over the transistors. The re-characterized libraries excludes these larger gates, meaning the synthesis tools can not use these when compiling the behavioral model.

As discussed in section 5.2, to avoid as much leakage current as possible, the ASIC will be clocked with the fastest clock the critical path allows. Using the standard cell libraries, the shortest clock period was found to be 10ns, which is almost 30 times faster then what was found in (7.4). Theoretically, it is possible to reduce the power dissipation 30 times by lowering the clock frequency, as shown in (5.3). This is however not true since the leakage increases. It is better to lower the supply voltage, which forces one to use a slower clock frequency, resulting in even lower power usage, low leakage current, and a design that is not unnecessary fast.

The resources used during synthesis are the same as in Table 5.1: 1275 registers, 2 adders, 2 subtracters, 2 multipliers, and 117 comparators. Since the re-characterized libraries does not contain large gates, smaller gates are combined to make larger. The area needed for realizing the circuit can therefore be larger. However, most of the differences in circuit area depends on the timing constrains. With a fast clock, the synthesis tool will insert buffers in an effort to meet the timing constrains, resulting in a large combinatorial area. The estimated area for all the different combinations of cell libraries and supply voltages are listed in Table 7.1.

TABLE 7.1: Estimated area after synthesis. The area for I/O pads are excluded.

			Estimated area (mm^2)		
Library	**V_{DD}**	**t_{clk}**	**Comb.**	**Noncomb.**	**Total**
LPHVT	1.2V	10ns	0.0255	0.0159	0.0414
LPHVT re-char	0.6V	400ns	0.0734	0.0163	0.0897
	0.5V	2200ns	0.0579	0.0167	0.0746
	0.4V	20000ns	0.0538	0.0167	0.0705
LPSVT re-char	0.6V	100ns	0.0561	0.0159	0.0720
	0.5V	300ns	0.0951	0.0159	0.1110
	0.4V	1500ns	0.0462	0.0164	0.0625
LPLVT re-char	0.6V	25ns	0.0244	0.0159	0.0403
	0.5V	90ns	0.0476	0.0159	0.0635
	0.4V	350ns	0.0567	0.0168	0.0735

7.3 Placement and Routing

The place and route (PnR) of the synthesized netlist was performed in Cadence Encounter. When everything was routed without any violations, the routed netlist and timing/delay information were extracted to be used in post-layout simulation for verification and power estimation.

7.3.1 Standard Power Implementation

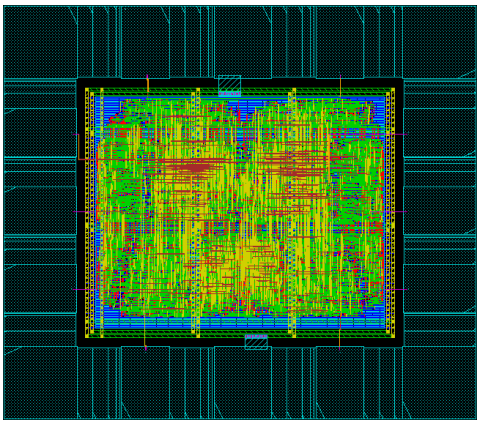


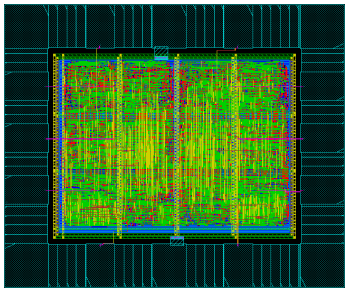
Fig. 7.1: ASIC layout from PnR using LPHVT libraries with 1.2V supply voltage. Core area is 0.072mm^2 $(300\mu\text{m} \times 240\mu\text{m})$. Core utilization is 72%.

7.3.2 Low Power Implementation

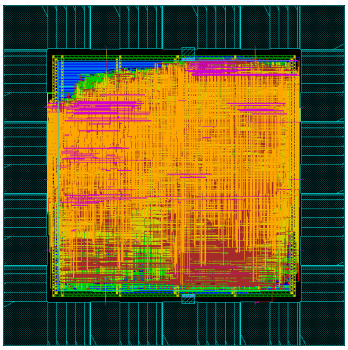
All the synthesized netlist using re-characterized cell libraries from Table 7.1 were placed and routed, except for the LPHVT 0.4V implementation. Because of the slow clock frequency required for the LPHVT 0.4V to function, more then 11 times slower then stated in (7.3), this implementation is ignored since it is not a feasible solution. The LPHVT 0.5V implementation also has a clock frequency slower than 560kHz, but it is not that far off and therefore included for comparison.

The size of the core area is set through trial-and-error, with the goal to utilize as much as possible of the available die area. If the size of the core is too large, the utilized area is small and the ASIC will be unnecessary expensive to manufacture. If the size of the core is too small, there will not be enough room for routing the metal connections between the transistors resulting in a non-functional ASIC. Looking at the layouts using the LPLVT libraries in Fig. 7.4, one can see that the utilization is very small compared to the other layouts in Fig. 7.1, 7.2, and 7.3. Unfortunately, a smaller core area introduced several violations and required

routing of metal wires outside the core. The different color scheme of the wires in Fig. 7.4 also indicates that more of the top metal layers are utilized in order to route all signals.

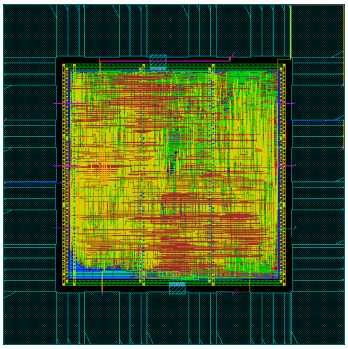


(a) LPHVT, $V_{\mathrm{DD}} = 0.6\mathrm{V}$. Core area is $0.12\mathrm{mm}^2$ $(400\mu\mathrm{m} \times 300\mu\mathrm{m})$. Core utilization is 71%.

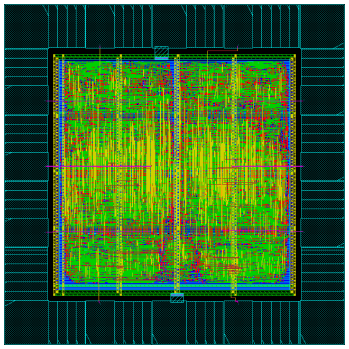


(b) LPHVT, $V_{\mathrm{DD}} = 0.5\mathrm{V}$. Core area is $0.16\mathrm{mm}^2$ $(400\mu\mathrm{m} \times 400\mu\mathrm{m})$. Core utilization is 75%.

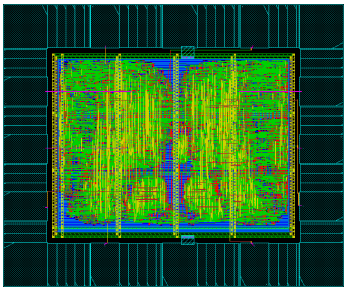
Fig. 7.2: ASIC layout from PnR using LPHVT re-characterized libraries.



(a) LPSVT, $V_{\mathrm{DD}} = 0.6\mathrm{V}$. Core area is $0.09\mathrm{mm}^2$ $(300\mu\mathrm{m} \times 300\mu\mathrm{m})$. Core utilization is 83%.

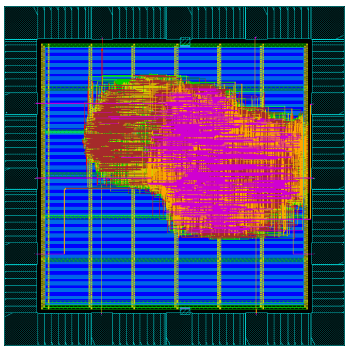


(b) LPSVT, $V_{\mathrm{DD}} = 0.5\mathrm{V}$. Core area is $0.16\mathrm{mm}^2$ $(400\mu\mathrm{m} \times 400\mu\mathrm{m})$. Core utilization is 64%.

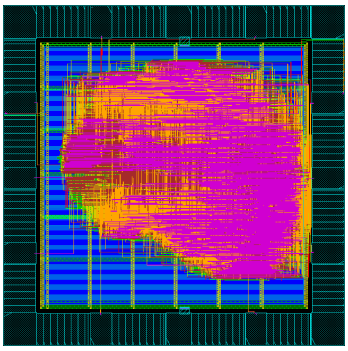


(c) LPSVT, $V_{\mathrm{DD}} = 0.4\mathrm{V}$. Core area is $0.12\mathrm{mm}^2$ $(400\mu\mathrm{m} \times 300\mu\mathrm{m})$. Core utilization is 75%.

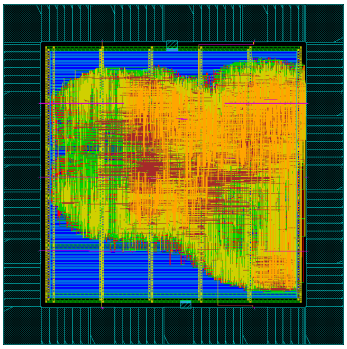
Fig. 7.3: ASIC layout from PnR using LPSVT re-characterized libraries.



(a) LPLVT, $V_{\mathrm{DD}} = 0.6\mathrm{V}$. Core area is $0.36\mathrm{mm}^2$ $(600\mu\mathrm{m} \times 600\mu\mathrm{m})$. Core utilization is 18%.



(b) LPLVT, $V_{\mathrm{DD}} = 0.5\mathrm{V}$. Core area is $0.36\mathrm{mm}^2$ $(600\mu\mathrm{m} \times 600\mu\mathrm{m})$. Core utilization is 23%.



(c) LPLVT, $V_{\mathrm{DD}} = 0.4\mathrm{V}$. Core area is $0.25\mathrm{mm}^2$ $(500\mu\mathrm{m} \times 500\mu\mathrm{m})$. Core utilization is 29%.

Fig. 7.4: ASIC layout from PnR using LPLVT re-characterized libraries.

7.4 Power Analysis

The power analysis is performed in *Synopsis Primetime* using signal activity information extracted from post-layout simulations in *QuestaSim*. All simulations was verified with the same testbench running at the corresponding clock frequency for each ASIC implementation. The number of samples that was run through each simulation was the same, meaning the slower the clock frequency, the longer the simulation time.

7.4.1 Standard Power Implementation

The results of the power analysis, using the standard 1.2V low power, high V_T (LPHVT) cell libraries, are listed in Table 7.2. The power dissipation of the I/O pads are excluded.

TABLE 7.2: Results of Primetime's power analysis for LPHVT cell library, $V_{DD} = 1.2V$, simulation time = 1ms. Clock constrains are listed in Table 7.3.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	1.169e-03	4.636e-04	2.673e-09	1.632e-03
register	2.843e-05	1.529e-05	8.962e-08	4.381e-05
combinational	9.124e-05	9.797e-05	1.049e-07	1.893e-04
Leakage percentage		0.01%		
Total Power		**1.865e-03**		

TABLE 7.3: Clock constrains for ASIC with standard 1.2V LPHVT cell library.

Constrain	**Time (ns)**
Clock period	10
Clock uncertainty	0.1
Clock transition, rise & fall	0.1
Input delay	0.75
Output delay	0.25

7.4.2 Low Power Implementation

The results of the power analysis, using re-characterized cell libraries with different threshold voltages, are listed in Table 7.4, 7.6, 7.8, 7.10, 7.12, 7.14, 7.16, and 7.18. The power dissipation of the I/O pads are excluded.

TABLE 7.4: Results of Primetime's power analysis for LPHVT re-characterized cell library, $V_{\mathrm{DD}} = 0.6\mathrm{V}$, simulation time = 40ms. Clock constrains are listed in Table 7.5.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	6.617e-06	0	0	6.617e-06
register	1.340e-07	1.942e-07	1.061e-08	3.388e-07
combinational	1.796e-06	1.184e-06	4.218e-08	3.022e-06
Leakage percentage		0.27%		
Total Power		**9.977e-06**		

TABLE 7.5: Clock constrains for ASIC with re-characterized 0.6V LPHVT cell library.

Constrain	**Time (ns)**
Clock period	400
Clock uncertainty	5
Clock transition, rise & fall	0.1
Input delay	20
Output delay	0.1

TABLE 7.6: Results of Primetime's power analysis for LPHVT re-characterized cell library, $V_{DD} = 0.5V$, simulation time = 220ms. Clock constrains are listed in Table 7.7.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	8.133e-07	0	0	8.133e-07
register	1.808e-08	1.343e-08	7.553e-09	3.906e-08
combinational	3.542e-07	1.561e-07	5.702e-08	5.674e-07
Leakage percentage		4.55%		
Total Power		**1.420e-06**		

TABLE 7.7: Clock constrains for ASIC with re-characterized 0.5V LPHVT cell library.

Constrain	**Time (ns)**
Clock period	2200
Clock uncertainty	10
Clock transition, rise & fall	0.1
Input delay	20
Output delay	0.1

TABLE 7.8: Results of Primetime's power analysis for LPSVT re-characterized cell library, $V_{DD} = 0.6V$, simulation time = 10ms. Clock constrains are listed in Table 7.9.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	2.630e-05	0	0	2.630e-05
register	5.033e-07	2.322e-07	1.998e-07	9.353e-07
combinational	5.706e-06	5.214e-06	4.888e-07	1.141e-05
Leakage percentage		1.78%		
Total Power		**3.864e-05**		

TABLE 7.9: Clock constrains for ASIC with re-characterized 0.6V LPSVT cell library.

Constrain	Time (ns)
Clock period	100
Clock uncertainty	5
Clock transition, rise & fall	0.1
Input delay	20
Output delay	0.1

TABLE 7.10: Results of Primetime's power analysis for LPSVT re-characterized cell library, $V_{DD} = 0.5$V, simulation time = 30ms. Clock constrains are listed in Table 7.11.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	6.117e-06	0	0	6.117e-06
register	1.156e-07	1.655e-07	1.368e-07	4.178e-07
combinational	2.108e-06	1.287e-06	8.912e-07	4.286e-06
Leakage percentage		9.50%		
Total Power		**1.082e-05**		

TABLE 7.11: Clock constrains for ASIC with re-characterized 0.5V LPSVT cell library.

Constrain	Time (ns)
Clock period	300
Clock uncertainty	5
Clock transition, rise & fall	0.1
Input delay	20
Output delay	0.1

TABLE 7.12: Results of Primetime's power analysis for LPSVT re-characterized cell library, $V_{DD} = 0.4V$, simulation time = 150ms. Clock constrains are listed in Table 7.13.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	7.807e-07	0	0	7.807e-07
register	1.609e-08	2.033e-08	8.921e-08	1.256e-07
combinational	2.261e-07	1.673e-07	4.194e-07	8.127e-07
Leakage percentage		29.59%		
Total Power		**1.719e-06**		

TABLE 7.13: Clock constrains for ASIC with re-characterized 0.4V LPSVT cell library.

Constrain	**Time (ns)**
Clock period	1500
Clock uncertainty	10
Clock transition, rise & fall	0.1
Input delay	20
Output delay	0.1

TABLE 7.14: Results of Primetime's power analysis for LPLVT re-characterized cell library, $V_{DD} = 0.6V$, simulation time = 2.5ms. Clock constrains are listed in Table 7.15.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	1.335e-04	7.180e-05	2.402e-07	2.056e-04
register	2.951e-06	2.820e-06	1.593e-06	7.363e-06
combinational	1.582e-05	1.730e-05	3.910e-06	3.703e-05
Leakage percentage		2.30%		
Total Power		**2.499e-04**		

TABLE 7.15: Clock constrains for ASIC with re-characterized 0.6V LPLVT cell library.

Constrain	Time (ns)
Clock period	25
Clock uncertainty	0.5
Clock transition, rise & fall	0.1
Input delay	1.5
Output delay	0.1

TABLE 7.16: Results of Primetime's power analysis for LPLVT re-characterized cell library, $V_{DD} = 0.5$V, simulation time = 9ms. Clock constrains are listed in Table 7.17.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	2.013e-05	0	0	2.013e-05
register	5.125e-07	5.542e-07	9.411e-07	2.008e-06
combinational	5.840e-06	5.902e-06	4.927e-06	1.667e-05
Leakage percentage		15.12%		
Total Power		**3.881e-05**		

TABLE 7.17: Clock constrains for ASIC with re-characterized 0.5V LPLVT cell library.

Constrain	Time (ns)
Clock period	90
Clock uncertainty	5
Clock transition, rise & fall	0.1
Input delay	10
Output delay	0.1

TABLE 7.18: Results of Primetime's power analysis for LPLVT re-characterized cell library, $V_{DD} = 0.4V$, simulation time = 35ms. Clock constrains are listed in Table 7.19.

	Power (W)			
	Internal	**Switching**	**Leakage**	**Total**
clock_network	3.356e-06	0	0	3.356e-06
register	9.297e-08	9.632e-08	5.885e-07	7.778e-07
combinational	8.957e-07	6.236e-07	2.781e-06	4.300e-06
Leakage percentage		39.96%		
Total Power		**8.430e-06**		

TABLE 7.19: Clock constrains for ASIC with re-characterized 0.4V LPLVT cell library.

Constrain	**Time (ns)**
Clock period	350
Clock uncertainty	5
Clock transition, rise & fall	0.1
Input delay	10
Output delay	0.1

The power usage of the different libraries is not the best way to find the best combination of V_{DD} and V_T, since they operate at different clock frequencies. As seen in (5.3), the clock frequency is proportional to the power usage. A fairer comparison is to divide the total power usage with the clock frequency, thus calculating the energy usage per clock cycle:

$$E = \frac{P}{f_{clk}} = \left[f_{clk} = \frac{1}{t_{clk}} \right] = P \cdot t_{clk}. \tag{7.5}$$

Results of using (7.5) on the different implementations are shown in Table 7.20. The total energy dissipation from the simulation-results are plotted in Fig. 7.5 and the leakage energy for each implementation is plotted in Fig. 7.6. The extrapolated data in Fig. 7.5 and Fig. 7.6 are approximated using second order polynomials.

TABLE 7.20: Power and energy dissipation of the different ASIC implementations.

Library, V_{DD}	**t_{clk} (ns)**	**P_{tot} (μW)**	**E_{tot} (pJ)**
LPHVT, 1.2V	10	1865	18.65
LPHVT, 0.6V	400	9.977	3.991
LPHVT, 0.5V	2200	1.420	3.124
LPSVT, 0.6V	100	38.64	3.846
LPSVT, 0.5V	300	10.82	3.246
LPSVT, 0.4V	1500	1.719	2.579
LPLVT, 0.6V	25	249.9	6.248
LPLVT, 0.5V	90	38.81	3.493
LPLVT, 0.4V	350	8.430	2.951

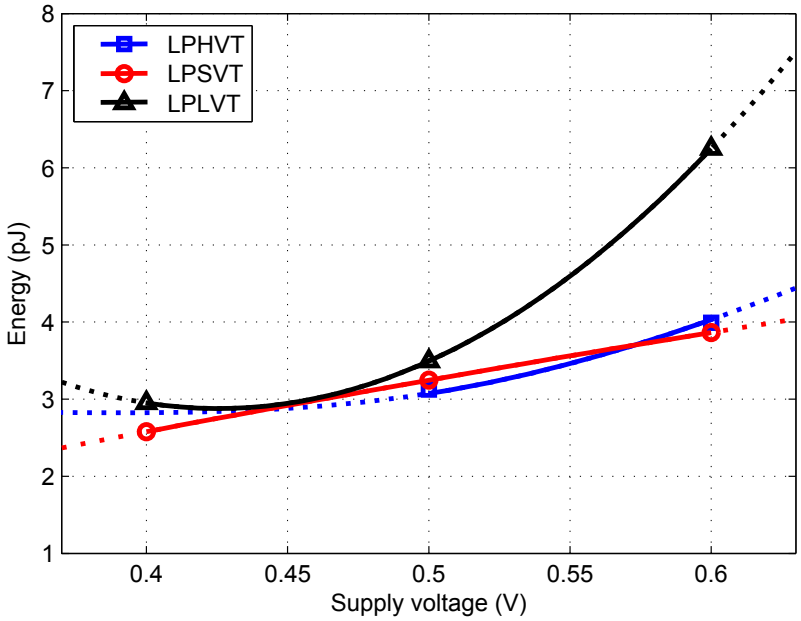


Fig. 7.5: Energy usage for different libraries and supply voltages.

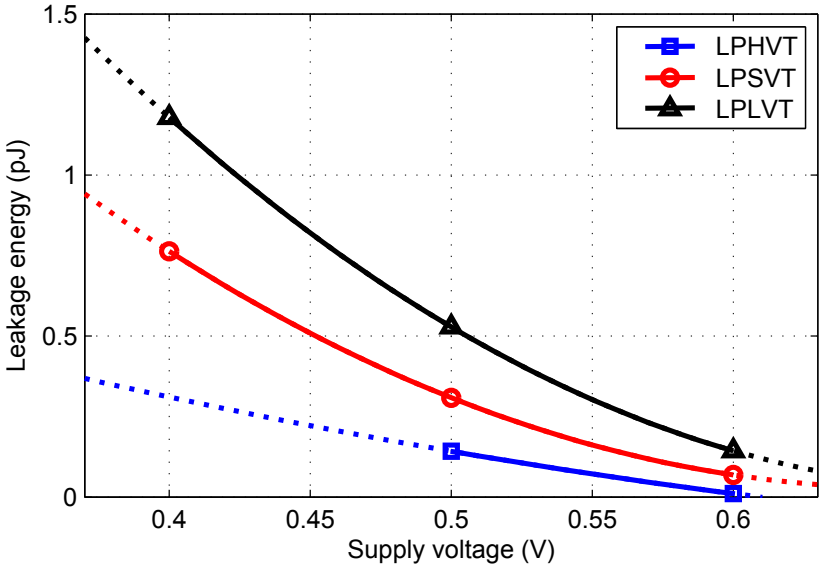


Fig. 7.6: Leakage energy for different libraries and supply voltages.

Chapter 8

Verification and Results

This chapter lists the equipment, apart from the FPGA previously described in section 6.1, used for measurements in the anechoic chamber and then gives the results from the measurements.

8.1 Equipment for Verification

8.1.1 Hearing Protectors

The hearing protectors used for testing the implemented AGC algorithm are called Peltor Tactical 7-SH (type MT1H7A-07). They have a cable with a standard 6.3mm stereo headphone male connector for playing back sounds in the internal speakers from an external source. With a 6.3mm-female-to-3.5mm-male-adapter, the speakers can be connected to the FPGA. However, the 50mW headphone amplifier on the FPGA proved not to be powerful enough to drive the speakers, so an external amplifier between the FPGA and the hearing protectors is needed. The microphone-wires from the two microphones on the hearing protectors are disconnected from the internal circuit boards and soldered to an external cable with a 3.5mm stereo headphone male connector in the other end.

For comparing the hardware implementation with a commercial product, another similar pair of hearing protectors called Peltor Tactical 7 Classic (type MT1H7A) were used. These are unmodified.

(a) Peltor Tactical 7 Classic. Unmodified.

(b) Peltor Tactical 7-SH. Modified with microphone-connector.

Fig. 8.1: Hearing protectors used for measurements.

8.1.2 Loudspeakers

Two different types of speakers where used during testing and verification. To generate white noise, the Norsonic Dodecahedron Loudspeakers Nor270H, Fig 8.2a, powered by the Norsonic Power Amplifier Nor280, Fig. 8.2b, was used.

The Fostex 6301B, Fig. 8.3, were used as amplifiers to drive the speakers in the hearing protectors. Two of these speakers were used, one for each stereo channel.

8.1.3 Head and Torso Simulator

A human analog is needed to measure the perceived noise from wearing or not wearing the hearing protectors. The Head and Torso Simulator Type 4128-C, Fig. 8.4a, from Brüel & Kjær is the size of an average adult and has microphones for ears. These microphones are connected to the Brüel & Kjær NEXUS Microphone Conditioner, Fig. 8.4b, which in turn is connected the the sound card of a computer for recording the noise picked up by the microphones.

8.1.4 Sound Level Meter

To get accurate results and for calibrating the gain LUT, one must know the true noise level during testing. The 01dB SdB+ Sound level meter, Fig. 8.5, is used to measure the true noise level next to the Head and Torso's ears.

(a) Norsonic Dodecahedron Loudspeaker Nor270H.

(b) Norsonic Power Amplifier Nor280.

Fig. 8.2: Norsonic dodecahedron loudspeaker and power amplifier. Used for noise generation. [12]

Fig. 8.3: Fostex 6301B Analog Personal Monitors. Used for driving the speakers in Peltor Tactical 7-SH hearing protectors. [13]

(a) Brüel & Kjær Head and Torso Simulator type 4128-C. [14]

(b) Brüel & Kjær NEXUS Microphone Conditioner type 2690. [15]

Fig. 8.4: Brüel & Kjær Head and Torso Simulator and NEXUS Microphone Conditioner. Used for recording noise with and without hearing protectors.

Fig. 8.5: 01dB SdB+ Sound level meter. Used for measuring the true noise level.

8.1.5 External Sound Card

To avoid noise caused by interference from components inside the computer during measurement, an external Roland UA-1EX USB audio interface sound card, Fig. 8.6, is used to record the sound picked up by the microphones in the Head and Torso Simulator.

Fig. 8.6: Roland UA-1EX USB audio interface sound card. Used to record the sounds heard by the Head and Torso Simulator. [16]

8.2 Noise Attenuation Measurement

When the implementation of the filters and AGC is complete, the most suitable LUT with the gain values has to be decided. Several different LUTs with different offsets, aforementioned in section 2.4 and 3.4, was tested in order to dial in the LUT that met the requirements. If a LUT dampens too much or dampens on unnecessary low decibel level, the audio quality will suffer.

The Norsonic Nor270H speaker was set up to play white noise at a specific level. The noise level was then increased with about 5dB for each of the following measurements. The reference noise level was measured just next to the Head and Torso's ear with the sound level meter. The input volume on the UA-1EX sound card was then calibrated to match that noise level with no hearing protectors on. Both pair of hearing protectors were put on the Head and Torso, one at a time, for comparison and the result is presented in Fig. 8.7.

The lookup table that dampened just enough to keep the noise level below 82dB is used for the measurements in Fig. 8.7, and is also the one seen in Fig. 3.7.

As seen in Fig. 8.7, the measured noise level for the AGC implementation done in this thesis peaks just below 82dB when the surrounding noise is around 85dB and then decreases slightly to around 80dB for higher noise levels. One can also see that the reference hearing protectors have a high amplification at lower noise levels, compared to the implementation done in this thesis. At approximately 80dB, both designs are pretty similar. The closer to ideal curve accomplished in

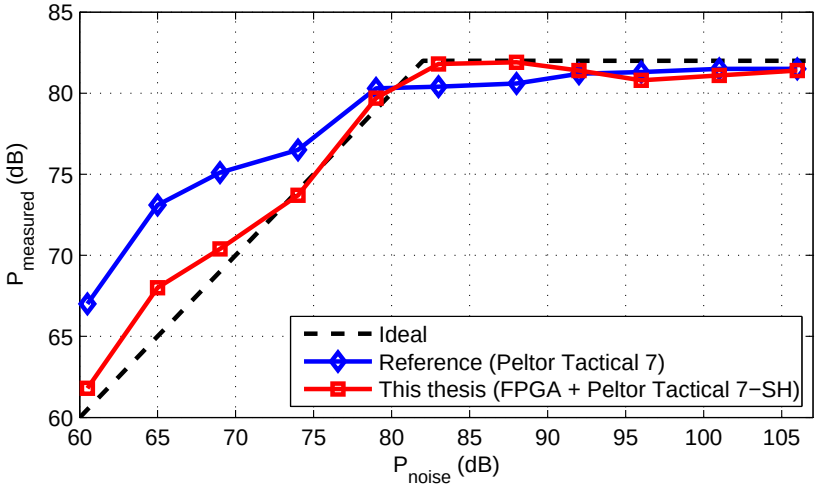


Fig. 8.7: Results from the dampening measurements in the anechoic chamber for the reference hearing protectors and the implementation of this thesis. The ideal curve has a gain $= 1$ up to 82dB before dampening, as in (3.9). P_{noise} is the noise level in the room and $P_{measured}$ is the noise level inside the hearing protectors.

this thesis allows for a more natural sound volume at harmless noise levels. This becomes clear when plotting the difference between the measured noise level to the ideal curve, as in Fig. 8.8.

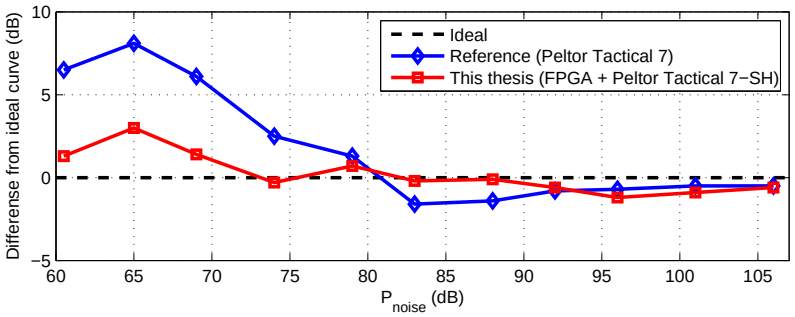


Fig. 8.8: The difference from the ideal curve in Fig. 8.7 for the reference hearing protectors and for the implementation of this thesis. P_{noise}, on the x-axis, is the noise level in the room and the y-axis shows the difference of the measured noise level inside the hearing protectors from the ideal curve, i.e. $P_{measured} - P_{ideal}$.

Chapter 9

Conclusions and Further Development

9.1 Conclusions

The aim for this thesis has been to implement a working AGC algorithm in hardware with focus on low power consumption. Measures, like resource sharing, has been taken to keep the area small. The comparison against an equivalent commercial product proves that the design is working. By implementing the same algorithm on an ASIC using re-characterized cell libraries for low power consumption, both the main goals for this thesis has been accomplished. Instead of running the ASIC with a standard supply voltage of 1.2V, a scaled down supply voltage in combination with a slower clock frequency results in a simulated power reduction of up to 86%.

The best compromise between speed and low power dissipation looks to be when using the LPSVT re-characterized cell libraries. LPHVT offered lower leakage with similar energy usage, but with a clock frequency that is too slow for this algorithm when V_{DD} is below 0.6V. The LPLVT libraries have the capability to operate much faster, but unless one is able to utilize more of the core area during PnR, this implementations are not a reasonable option. The ASIC with the lowest simulated energy usage, LPSVT with $V_{DD} = 0.4$V, dissipates 2.58pJ per clock cycle at 667kHz and can handle a ADC sample rate of 9.5kHz. If one want an implementation capable of higher sample rate, switching up to $V_{DD} = 0.5$V yields a higher theoretical audio quality than a studio recorded CD (44.1kHz), with a energy dissipation of 3.25pJ per clock cycle.

Since the gain factor is 1 for harmless noise levels, the implementation in this thesis has a very natural sound level. An amplification for low sound levels, found in the reference hearing protectors for example, have the advantage of emphasizing speech for instance. However, it also amplifies unwanted sounds, like distant traffic noise or the sound of the user's own footsteps. The equal attenuation for both ears also results in a more pleasing listening experience. Without the independent AGCs for each ear, the annoying swaying effect mentioned in the introduction is eliminated.

9.2 Future Work

For the proposed solution in this thesis, 16-bit resolution for the sample were chosen to have the best audio quality the codec can deliver. With fewer bits, the audio quality will suffer, but the power consumption and chip area would also decrease. With more time, next step in the optimization would be to find the number of bits that suffice for an acceptable audio quality and see what power usage and area that yields.

To achieve a more accurate power estimation for the AGC, the samples should be A-weighted as discussed in section 2.4. An additional filter will increase the number of clock cycles needed to process one sample, and will require a slightly higher clock frequency to continue to have the same sample rate on the ADC and DAC.

The proposed solution does not work by itself. If it were to be fabricated and put in a commercial product, the circuit should contain its own ADC and DAC. By embedding an ADC and a DAC on the ASIC, the need for serial data transfer is also removed. Sending the samples in parallel from/to the ADC/DAC almost halves the time the algorithm takes to process one sample, allowing for even lower power dissipation.

Having the LUT hard-coded on the ASIC is very area- and power efficient, but can be a bit limiting. If the LUT were to consist of an external interchangeable or reprogrammable ROM, it would allow for more and different applications. Some might want a bit of amplification at lower audio levels to enhance their hearing, or people with a sensitive hearing might require a lower threshold level than 82dB. By just changing the values in the LUT, the AGC can still be used without any alterations.

References

[1] Bengt Johansson. *Buller och Bullerbekämpning*. "Arbetsmiljöverket" (Swedish Work Environment Authority), 4th edition, 2002.

[2] Maziar Soltani and Anna Lilja-Ramusson. Signal processing for hearing protectors. Master's thesis, Blekinge Institute of Technology, Sweden, 2006.

[3] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2005.

[4] Jonas Skeppstedt and Christian Söderberg. *Writing Efficient C Code: A Thorough Introduction for Java Programmers*. Skeppberg AB, 1st edition, 2013.

[5] Pong P. Chu. *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. Wiley-IEEE Press, 2006.

[6] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nicolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2nd edition, 2003.

[7] Xilinx. "ML505/ML506/ML507 Evaluation Platform", User Guide, version 3.1.2, May 16, 2011.

[8] Analog Devices. "AC'97 SoundMAX Codec", AD1981B datasheet, Rev. C, 2005.

[9] Xilinx. XUPV5-LX110T. `http://www.xilinx.com/univ/xupv5-lx110t.htm`. Accessed: 2015-07-30.

[10] John Terragnoli. ECE383_Lab02, GitHub repository, "`ac97.vhd`" (2015). `https://github.com/JohnTerragnoli/ECE383_Lab02/blob/master/Code/ac97.vhd`. Accessed: 2015-03-20.

[11] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Prentice Hall, 4th edition, 2006.

[12] Norsonic. Noise Excitation for Building Acoustics. `http://www.campbell-associates.co.uk/products/Norsonic/productdata/071005%20Noise%20Excitation%20Kit%20issue%20201007v2.pdf`. Accessed: 2015-07-27.

[13] Fostex. 6301 Analogue Personal Monitors. http://www.fostexinternational.com/docs/products/6301_Analogue.shtml. Accessed: 2015-07-27.

[14] Brüel & Kjær. Head And Torso Simulator 4128-C. http://www.bksv.com/Products/transducers/ear-simulators/head-and-torso/hats-type-4128c. Accessed: 2015-07-27.

[15] Brüel & Kjær. NEXUS Microphone Conditioner - Type 2690-A. http://www.bksv.com/Products/transducers/conditioning/microphone/2690A0F2. Accessed: 2015-07-27.

[16] Roland. UA-1EX USB audio interface sound card. http://www.roland.com/products/ua-1ex/support/. Accessed: 2015-07-23.