FPGA Implementation of an AC3 Decoder

Dapeng Han
Abstract

The aim of this thesis is to explore the possibility of integrating an AC3 audio decoding module into the company’s current product. Due to limited left resources on the FPGA chip in the company’s current product, the focus of this thesis is to be resource efficient. In this thesis, a system for AC3 audio decoding is designed and implemented. In order to use less logic on FPGA, PicoBlaze soft processor is used to control the whole processing flow. The system is designed and synthesized for a Spartan-6 FPGA which can be easily ported to the company’s current platform.
Acknowledgments

First, I would like to thank Pär Gunnars Risberg for him giving me this thesis opportunity in Opalum AB. He has given me encouragement and support in difficult moments.

I would like thank Erik Lindahl for the help with all the technical problems and patience of answering my questions.

Thank you to all the other employees in Opalum AB for their support and encouragement, sharing their knowledge and experience and being great friends.

Thank you to my examiner Kent Palmkvist for making this thesis possible.

Thank you to my friend and thesis opponent Oscar Kleback for his encouragement and patience.

Last but not least, I would like to thank all my friends during my study period in Sweden for their support and company.

Oslo, April 2017
Dapeng Han
# Contents

<table>
<thead>
<tr>
<th>Notation</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Problems</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Previous Work</td>
<td>2</td>
</tr>
<tr>
<td>1.5 Limitations</td>
<td>3</td>
</tr>
<tr>
<td>1.6 Outline</td>
<td>3</td>
</tr>
<tr>
<td>2 Dolby Digital Standard</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Basic Theories</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Psychoacoustic Audio</td>
<td>5</td>
</tr>
<tr>
<td>2.1.2 Masking</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 Pulse Code Modulation</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4 Number Representation</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Overview</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Dolby Digital Encoding</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Dolby Digital Decoding</td>
<td>11</td>
</tr>
<tr>
<td>2.5 Dolby Digital Frame Structure</td>
<td>12</td>
</tr>
<tr>
<td>3 Dolby Digital Decoding Process</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>15</td>
</tr>
<tr>
<td>3.1.1 Decoding Flow</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Key Steps</td>
<td>18</td>
</tr>
<tr>
<td>3.2.1 Synchronization and Error Detection</td>
<td>18</td>
</tr>
<tr>
<td>3.2.2 Exponents Decoding</td>
<td>19</td>
</tr>
<tr>
<td>3.2.3 Bit Allocation</td>
<td>21</td>
</tr>
<tr>
<td>3.2.4 Decoding Mantissas</td>
<td>23</td>
</tr>
<tr>
<td>3.2.5 Channel Coupling</td>
<td>25</td>
</tr>
<tr>
<td>3.2.6 Inverse Transform</td>
<td>25</td>
</tr>
<tr>
<td>4 Hardware Platform</td>
<td>27</td>
</tr>
</tbody>
</table>
# Contents

4.1 Introduction ............................................ 27  
4.2 Xilinx SP601 Evaluation Board ............................. 27  
4.3 Xilinx ISE ................................................ 27  
4.4 PicoBlaze Micro Controller ............................... 29  
  4.4.1 Introduction ......................................... 29  
  4.4.2 Architecture .......................................... 29  
  4.4.3 Components and Connections ......................... 29  
  4.4.4 Program Environment ................................. 30  

5 Implementation ............................................. 33  
  5.1 Propose .................................................. 33  
  5.2 Data Flow ............................................... 33  
  5.3 Implementation ......................................... 34  
    5.3.1 PicoBlaze ........................................ 34  
    5.3.2 Exponent Decoder .................................. 36  
    5.3.3 Bit Allocation ..................................... 40  
    5.3.4 Mantissa Ungroup and Mantissa Decoder ........... 50  
    5.3.5 IMDCT ............................................. 51  

6 Results and Conclusion .................................... 55  
  6.1 Results ................................................. 55  
  6.2 Conclusion .............................................. 56  
  6.3 Future Work ............................................ 56  

Bibliography .................................................. 59
### Notation

#### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUX</td>
<td>auxiliary data</td>
</tr>
<tr>
<td>BRAM</td>
<td>block random-access memory</td>
</tr>
<tr>
<td>BSI</td>
<td>bit stream information</td>
</tr>
<tr>
<td>CRC</td>
<td>cyclic redundancy check</td>
</tr>
<tr>
<td>DVD</td>
<td>digital versatile disc</td>
</tr>
<tr>
<td>FBW</td>
<td>full bandwidth</td>
</tr>
<tr>
<td>FIFO</td>
<td>first in first out</td>
</tr>
<tr>
<td>FPGA</td>
<td>field-programmable gate array</td>
</tr>
<tr>
<td>HDTV</td>
<td>high definition television</td>
</tr>
<tr>
<td>IMDCT</td>
<td>inverse modified discrete cosine transform</td>
</tr>
<tr>
<td>LFE</td>
<td>low frequency effects</td>
</tr>
<tr>
<td>MCU</td>
<td>microcontroller unit</td>
</tr>
<tr>
<td>PCM</td>
<td>pulse-code modulation</td>
</tr>
</tbody>
</table>
1

Introduction

1.1 Background

Dolby Laboratories developed the audio compression technology called Dolby Digital in 1990s. This technology makes storing and transmitting digital audio more efficiently possible. It is also known as AC3. Nowadays it is widely used in the audio industry all over the world.

![Dolby Digital Logo](image)

Figure 1.1: Dolby Digital Logo [8]

Dolby Digital is first used in the Movie Batman Returns. So far, Dolby Digital has been the true standard for movie distribution and cinema equipment. Besides the movie industry, Dolby Digital can also be used in family environment. It is the audio standard for all DVDs and HDTV in North America. Even a lot of video games support Dolby Digital as a feature.

1.2 Objectives

The main products of Opalum is speakers. It is commonly accepted in the sound industry that excellent sound requires large loudspeakers. Opalum decided to challenge the veracity of this and successfully developed a technology which unites excellent, resonant sound with slim design. [11]

The control hub is the center of all Opalum products. It let’s you integrate with a variety of sources such as TV’s, CD-players, DVD-players, Apple TV’s,
Airport Express’ and so on [11]. The hub transmits the digital audio to speakers.

1.3 Problems

The control hub does not support Dolby Digital decoding now. If the digital audio source is Dolby Digital, the speakers are just mute. To solve this problem, the company wants to explore the possibility to build a Dolby Digital decoder in FPGA due to the fact that there is a tiny FPGA inside the product already. If AC3 decoding can be done in the current FPGA, it will save the cost for the company.

One of the problems is that the resource in the tiny FPGA is limited and the Dolby Digital has a fairly complex structure, so it is to be investigated whether a Dolby Digital decoder can fit into the FPGA or not.

Another problem is that the AC3 frame has a tricky structure which the former information bits decide if the following information bits exist or not. This requires a flexible design to handle the frame structure.

1.4 Previous Work

There are several other approaches to decode AC3 can be found online.

- In reference [10], it is a hardware/software co-design solution based an ARM platform. It performs most the decoding process in software, such as data parsing, exponent decoding, bit allocation. But it design dedicated hardware for IMDCT calculation, because there are many floating point calculation while the ARM platform does not provide floating point calculation instructions. If the IMDCT is done in software, it will have a bad perform. It is a similar approach to this thesis.

- In reference [5], it is describes how to install and work with Texas Instruments (TI) Dolby AC3 Version 3 Decoder implementation on the C64x+ platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component. It is a software solution which can be fast integrated if the Texas Instruments platform are used.

- In reference [12], it describes the design and implementation of the AC3 decoder based on an audio specific DSP core. The work includes the DSP core design and software development. It is also a software solution with specific designed hardware.

- In reference [2], this paper presented an SOC based HW/SW co-design architecture for multi-standard audio decoding. It is developed to support the audio standards of AAC LC profile, Dolby AC3, Ogg Vorbis, MPEG-1 Layer 3 (MP3) and Windows Media Audio (WMA). A VLSI reconfigurable filterbank based on CORDIC algorithm is developed to accelerate the multi standard decoding process. The architecture is also flexible for extending support of new formats and standards.
From the work mentioned above, it is a common opinion that the AC3 decoding is better to be done with a hardware/software co-design solution. So this thesis is following the same principle.

1.5 Limitations

For this thesis project, ‘ATSC Standard: Digital Audio Compression (AC-3, E-AC-3)’ 2012 version is followed, which can be found in [1]. The target sample rate is 48kHz. Due to limited time and the request from the company, some decoding steps are not considered because they are not mandatory to the core function of decoding process. For example, Rematrixing, Dynamic Range Compression and Downmix are not implemented. Also, only uncoupling channels are considered.

1.6 Outline

- Chapter 2 and 3 explain the Dolby Digital standard and the Dolby Digital frame structure in detail and cover the decoding process.

- Chapter 4 talks about the hardware platform used in the project and PicoBlaze micro controller which is a key component in decoding.

- Chapter 5 proposes a design solution for Dolby Digital decoding in FPGA by using the hardware in Chapter 4 and talks about implementation details and simulation results.

- Chapter 6 gives a short summary about the project and proposals for future work.
2.1 Basic Theories

2.1.1 Psychoacoustic Audio

Human’s hearing system is a very complex system. The frequency range which human ear can hear is from 20Hz to 20kHz. But the sensitivity of the human ear depends on the frequency and human ears are most sensitive to the frequency range 2kHz to 4kHz. For every frequency, there is a sound pressure level which is a value meaning if the sound at this frequency can be sensed by human ears or not. This sound pressure level can be calculated by the Equation 5.4.

\[
T_q(f) = 3.64 \left( \frac{f}{1000} \right)^{-0.8} + 6.5e^{-0.6(f/1000-3.3)^2} + 10^{-3} \left( \frac{f}{1000} \right)^2
\]  

(2.1)

where \( f \) is the frequency. This equation can be found in [6] and is characterized in Figure 2.1.

Sound with strength under the line can not be sensed by human ears. In this case, there is no need to transform this part of information. Therefore, this kind of information is removed from the signal in digital audio compression process.

2.1.2 Masking

Masking is another important concept in psychoacoustic audio. It refers to the case that a sound which can be heard by human ears are not sensible due to a stronger sound. This can happen in both frequency and time domain. Figure 2.2 illustrates a spectral masking case.

The bigger arrow represents a strong sound at 8kHz and this sound rises the curve from Figure 2.1. There is a weaker sound at a close frequency x and this
Figure 2.1: Sensible Sound Strength of Human Ears
[6]

Figure 2.2: Masking Example in Frequency Domain
[6]
sound is supposed to be sensed by human ears, but in this case, the sound at frequency x can not be heard due to the appearance of the stronger sound. So when the digital audio information is compressed, the sound information at frequency x can be deleted. Because this kind of sound can not be heard, there is no need to do audio coding on them.

2.1.3 Pulse Code Modulation

Pulse Code Modulation (PCM) is an important method to represent and transmit analogue signal digitally. In the encoder, the analogue signal are sampled at a constant time interval and every sample is quantized to a digital value. The quantized value is coded into a set of binary numbers. These binary number can be used for transmitting and signal processing. For general audio signal, the sample frequency is 44.1kHz and the quantization is 16 bits. From Nyquist sampling theorem, the signal with a bandwidth less than 22.05kHz can be rebuild. The bandwidth is a little larger than the range which human ears can hear. In quantization, different step sizes can be used to get a higher compression ratio.

![Figure 2.3: Sampling and Quantization of A Signal (red) for 4-bit PCM](image)

2.1.4 Number Representation

Two’s Complement

Two’s complement numbers are identical to unsigned binary numbers except that the most significant bit position has a weight of $-2^{N-1}$ instead of $2^{N-1}$. They overcome the shortcomings of sign/magnitude numbers: zero has a single representation, and ordinary addition works. In two’s complement representation, zero is written as all zeros: 00...000. The most positive number has a 0 in the most significant position and 1’s elsewhere: 01...111 = $2^{N-1} - 1$. The most negative number has a 1 in the most significant position and 0’s elsewhere: 10...000 = $-2^{N-1}$. And -1 is written as all ones: 11...111.[4]
Floating Point

Floating point represents numbers in which the binary point is not fixed, as it is for integers. Just as in scientific notation, numbers are represented as a single nonzero digit to the left of the binary point.

The representation of a floating-point number is shown in Figure 2.4 where $s$ is the sign of the floating-point number (1 meaning negative), exponent is the value of the exponent field and fraction.

![Figure 2.4: Floating Point](image)

In general, floating-point numbers are of the form

$$(-1)^S \times F \times 2^E$$

$F$ involves the value in the fraction field and $E$ involves the value in the exponent field[3].

### 2.2 Overview

By taking advantages of the phenomenon in previous section, a digital compression algorithm can be developed so that the total amount of information used to present the information can be reduced. The purpose of the digital compression algorithm is to use the least possible digital bits to store and transmit the audio more efficiently and at the same time the decode output sounds about the same quality as the original signal.

Dolby digital format is based on Psychoacoustics to improve the compression performance. Because human ear has different sensitivity to different frequencies, Dolby Digital can allocate different data amount based on the dynamic property of the signal. The signals with intensive spectrum and strength has more data to represent than other signals. The signals under masking take little or no data. And combined with coupling and rematrixing technology, the data stream can be further decreased.

Dolby Digital supports surround sound by five full range channels, left, right, left surround, right surround, and center, with a low frequency effects (LFE) channel. It is shown in Figure 2.5.

The full range channels cover the frequency band 20 - 20000 Hz and the low frequency effects channel covers the frequency band 20 - 120 Hz which is used to produce low bass effects.

The three front channels (Left, Center, and Right) provide crisp, clean dialogue and accurate placement of on-screen sounds. The twin surround channels
2.3 Dolby Digital Encoding

The Dolby Digital encoder takes PCM audio samples as input and outputs Dolby Digital bit stream. The encoding process is not required in this project. It is described here briefly.

The diagram of encoding is shown in Figure 2.7. The encoding process starts with transforming the PCM time samples into frequency coefficients in the analysis filter bank. Because of the overlapping step in the transforming, one input will be in two successive output blocks. Every frequency coefficient are represented by an exponent and an mantissa in binary notation. The exponents are encoded to represent the signal spectrum in the spectral envelope encoding block. The output is called spectral envelope and is used in the core bit allocation to

\[ 6 \text{ channels} \times 48k\text{Hz} \times 18\text{bits} = 5,184\text{Mbps} \]  

After encoding in Dolby Digital encoder, the data rate drops to 384 kbps. The satellite equipment transmits the data in radio frequency signals but the signal bandwidth and the power needed for transmission is reduced by more than a factor of 13 due to Dolby Digital audio compression algorithm. The signal is received in the receiver and decoded back to original audio signals after the decoder.
Figure 2.6: Example Application of Dolby Digital to Satellite Audio Transmission [1]

Figure 2.7: Dolby Digital Encoding
calculate how many bits to distribute to each mantissa. These exponents and mantissas data are included in every audio blocks. Six blocks are formatted into a Dolby Digital audio frame. The Dolby Digital bit stream consists of a sequence of Dolby Digital audio frames.

In reality, the encoding process has more details than the diagram.

- Two cyclic redundancy check (CRC) words are contained in the frame to verify if the frame is right. One is in the end of a frame. The other one is in the header of a frame with other information like bit rate, sample rate, number of channels, synchronize word, etc.

- Analysis filter bank block is dynamically reconfigurable for audio blocks with different time or frequency characteristics.

- Exponents are encoded in spectral envelope encoding block with different time and frequency resolution.

- The high frequency part of the channels can be coupled together to reduce the data amount.

### 2.4 Dolby Digital Decoding

The decoding process is basically the inverse of the encoding process. A diagram of decoding is shown in Figure 2.8.

![Figure 2.8: Dolby Digital Decoding](image)

First, the decoder has to synchronize to the encoded bit stream. Then the decoder checks if the frame is error free and if it is error free, the decoder will parse the frame into different types like the encoded spectral envelope, the quantized mantissas in the diagram and other side information. Then bit allocation block uses the encoded spectral envelope information to calculate the bit allocation information which is used in mantissa de-quantization block to generate mantissas.
Spectral envelope outputs exponents. By using exponents and mantissas, synthesis filter bank outputs PCM time samples. The decoding process will be described in more details in next chapter.

2.5 Dolby Digital Frame Structure

Dolby Digital bit stream consists of Dolby Digital frame sequence which is shown in Figure 2.9.

![Figure 2.9: Dolby Digital Bit Stream](image)

Each Dolby Digital frame can be divided into three parts, the frame head, six audio blocks and the frame end. The frame head contains synchronization information (SI) and bit stream information (BSI). The first 16 bits in SI are syncword, which is used to synchronize the Dolby Digital frame. It is always 0x0B77 or "0000 1011 0111 0111" in binary representation. There is a possibility that the data in the frame is also 0x0B77. The following 16 bits are a CRC word which is used to check if the synchronized frame is correct. Fscod indicates sample rate and Frmsizcod indicates frame size. Details about the meaning of their value can be found in [1]. The SI structure is shown in Figure 2.10.

![Figure 2.10: SI Syntax and Word Size](image)

BSI contains all the side information for the whole frame, for example if an audio reuses the previous information, if the a channel is in coupling, if the transform is 512 point transform, etc.

![Figure 2.11: Dolby Digital Frame Structure](image)
Frame structure is showed in Figure 2.11. Every frame has six audio blocks. Each block contains 256 samples for every channel. In every audio block, there are information about block switch, coupling coordinates, exponents, bit allocation information, mantissas. The data in audio block 0 can be reused in the other blocks, i.e. the data can be shared in the frame. The later audio blocks can reuse the previous data by setting certain positions. In this way, the data amount is reduced. But for decoder, the previous data must be stored for the following blocks. Audio Block Structure is shown in Figure 2.12.

<table>
<thead>
<tr>
<th>Block Switch and Dither flag</th>
<th>Dynamic Range Control</th>
<th>Coupling Info</th>
<th>Rematrix Info</th>
<th>Exponent Strategy</th>
<th>Exponents for Each Channel</th>
<th>Bit Allocation Info</th>
<th>Quantized Mantissas for Each Channel</th>
</tr>
</thead>
</table>

**Figure 2.12: Audio Block Structure**

In the frame end, there are auxiliary data (AUX) and a second CRC word. AUX data are unused data. This CRC word can be used for error detection.
3.1 Overview

The decoding process flow is shown in Figure 3.1. The input data stream can be from a transmission system such like Sony/Philips Digital Interface Format (SPDIF). The details about the transmission system is beyond the scope of this thesis. The input data can be as a constant data stream with a certain data rate or a large amount of data in a short time. Due to this, a buffer is necessary.

3.1.1 Decoding Flow

Synchronization and Error Detection

Dolby digital synchronization word is 0x0b77 which is a 16 bit wide word. In many kinds of transmission, dolby digital data are often transmitted by byte or 16 bit word alignment. This fact reduces the error possibility of synchronization of the data frame and simplified the dolby digital decoder.

Unpack BSI and Side Information

In a Dolby Digital frame, there are different kinds of side information contained before audio blocks. The side information data is used to control and offer parameters for the decoding process. It is applied to all the six blocks and every channel in every block.

Due to the fact that the side information data is used in the following process, the data must be parsed and stored in registers from the input buffer, so it can be used when required.
Figure 3.1: Dolby Digital Decoding Process
3.1 Overview

**Decode Exponents**

The exponents are encoded in a certain form in a Dolby Digital frame, so the encoded form must be known. The number of exponents are determined in different ways for different channels. There are always 7 exponents for LFE channel if this channel exists.

Second key factor in exponent decoding is the exponent strategy in every channel which is explained later.

**Bit Allocation**

Bit allocation step decides how many bits are distributed to each mantissa. It takes the coded exponents and outputs the bit allocation values \textit{baps} (will be explained in section 3.2.3) for each coded mantissa.

**Unpack, Ungroup, Dequantize Mantissas**

In the Dolby Digital frame, the encoded mantissas are continuous bit stream. The decoder should take each mantissa from the stream based on baps value, since the baps value decides how many bits every mantissa has. Some mantissas are grouped together to reduce the data amount. They need to be ungrouped in the decoder. Then each mantissa need to be dequantized according to corresponding exponent.

**Decoupling**

In encoding process, the high frequency signal in every channel are combined together in coupling channel. So during the decoding process, the decoupling step recover the high frequency signal of every coupled channel. Only uncoupling channels are considered in this thesis, so this step is not covered by this thesis.

**Rematrixing**

Rematrixing technique is only used in 2/0 mode i.e. only the left and right channels are in use. If it is applied, the data from the two channels are encoded into the sum and the difference of the two channels. In this way, Dolby Digital can reduce the data to transmit.

**Dynamic Range Compression**

During the decoding process, the decode can change the magnitude of the coefficients based on the control and side information. This step is not mandatory and it is not covered by this thesis.

**Inverse Transform**

This step transform the coefficients in frequency domain to time domain.
Window, Overlap/Add

This step windows the output from inverse transform. The two continuous data block should overlap and add half of each other to get the final PCM value.

Downmixing

Dolby Digital output 5.1 channels i.e. 5 FBW channels and 1 LFE channel. In some cases, the output channel number is less than the channel number in the data information. Downmixing is used for this kind of cases. It is not covered by this thesis.

PCM Output Buffer

The decoding process may not match the transmission, so an output buffer is needed.

Output PCM

The output PCM data can be used to feed into a digital to analogue converter (DAC) or other post signal processing which is covered in this thesis.

3.2 Key Steps

3.2.1 Synchronization and Error Detection

The synchronization word for Dolby Digital is 0x0B77. The decoder detects this word to decide if it receives a frame. This word can also be a information data inside of the frame, so there is a possibility of false detection. Without byte or word alignment, the value of error possibility is 19 percent per frame. With byte alignment, the value is 2.5 percent and with 16 bits word alignment, the value is 1.2 percent.

As it explained in last chapter, there are two CRC words contained in one frame. One is at the head of a frame and the other one is at the end. They are called CRC1 and CRC2 respectively.

CRC1 is used to check the first \( \frac{5}{8} \) of a frame. The result of CRC check can be available after the first \( \frac{5}{8} \) of a frame received. CRC2 is used to check the whole frame.

Even with CRC check, there is still a possibility of false synchronization. The value is reliable to 0.0015 percent [1].

Combining CRC check and byte alignment, the probability of false synchronization word detection drops to 0.000035 percent which can satisfy the requirement of most applications.
3.2.2 Exponents Decoding

Overview

Before the Inverse Transform step, the coefficients in time domain are represented in floating point form. In the data stream, the floating point data are separated and transmitted as exponents and mantissas. The exponents are encoded and packed in the Dolby Digital encoder. They are decoded in the following way.

Each Dolby Digital frame contains 6 audio blocks. The data may or may not be shared in the whole frame. So the exponents information in Audio Block 0 can be reused in the following audio blocks, Audio Block 1 to Audio Block 5. Therefore there will always be a new set of exponents for every channel - all the independent channels, all the coupled channels, the coupling channel and the low frequency effect channel.

The exponents are five bit wide data and they contain the information that how many zeroes in the front of the coefficients in frequency domain. The range of the exponents are from 0 to 24. If an exponent is 0, it means that the corresponding coefficient has no zeroes in the front. For the coefficients having more than 24 zeroes in the front, the exponent is 24.

Dolby Digital can reduce the data amount by using differential coding. In the FBW or LFE channel, the first exponent is a 4 bit wide value, so the range of this value is from 0 to 15. It contains the information about how many zeroes are in the front of the first coefficient. The following exponents information are sent as differential values. To get the real exponents, the decoder just needs to add the differential value to the previous exponent and this exponent value is used for next exponent calculation.

Another important technique in exponent encoding and decoding is called exponent strategy. By using this technique, the differential values of the exponents can be further packed into groups which further reduces the data amount.

There are three strategies. They are called D15, D25, and D45. For FBW and LFE channels, the 4 bit absolute exponent has a value range from 0 to 15. If the value is larger than 15, it is set as 15. For coupling channel, the absolute value is also 4 bit wide, but it represents a 5 bit value without least significant bit.

The exponent strategy coding in encoding process is not explained in this thesis. It can be found in [1].

Exponent Strategy Decoding

For FBW channels, the exponent strategy is determined by 2 bit data \( \text{chexpstr}[ch] \) and for coupling channel, it is determined by 2 bit data \( \text{cplexpstr} \). The meaning of the data are shown in Table 3.1.

For LFE channel, the exponent strategy is determined by \( \text{lfeexpstr} \). When \( \text{lfeexpstr} \) is '0', the decoder should reuse the prior exponents. When \( \text{lfeexpstr} \) is '1', the exponent strategy is D15.

To get the number of exponents in every channel, the channel bandwidth information is needed. For the coupled channels and the channels which are not in coupling, the starting mantissa bin number is 0. [1]
Table 3.1: Meaning of chexpstr[ch], cplexpstr [1]

<table>
<thead>
<tr>
<th>chexpstr[ch], cplexpstr</th>
<th>Exponent Strategy</th>
<th>Exponents per Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>reuse prior exponents</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>D15</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>D25</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>D45</td>
<td>12</td>
</tr>
</tbody>
</table>

For the channels which are not in coupling, the end mantissa bin number is

\[ endmant[ch] = ((chbwcod[ch] + 12) \times 3) + 37 \]  (3.1)

where chbwcod[ch] is Channel Bandwidth Code which is an unsigned integer which defines the upper band edge for fullbandwidth channel [ch]. [1]

For the coupled channels,

\[ endmant[ch] = cplstrtmant \]  (3.2)

where

\[ cplstrtmant = (cplbegf \times 12) + 37 \]  (3.3)

For the coupling channels, due to that the information in coupling channel represents the high frequency information in FBW channels, the starting mantissa bin number is not 0. The coupling channel starting and ending mantissa bins are defined as cplstrtmant and cplendmant. cplstrtmant can calculated in the above equation and

\[ cplstrtmant = ((cplendf + 3) \times 12) + 37 \]  (3.4)

where the value of cplbegf Coupling Begin Frequency Code and cplendf Coupling end Frequency Code can be found in the frame.

For LFE channel, if it is turned on, it always starts at 0 and ends at 7, due to it only represents the low frequency information.

For the FBW channels, the exponents structure in a Dolby Digital frame can be found in [1].

The group number of exponents in each channel is represented by nchgrps[ch]. It can be driven in the following equation.

\[ nchgrps[ch] = \text{truncate} \left( \left( \text{endmant[ch]} - 1 \right) / 3 \right); \text{forD15} \]
\[ = \text{truncate} \left( \left( \text{endmant[ch]} - 1 + 3 \right) / 6 \right); \text{forD25} \]  (3.5)
\[ = \text{truncate} \left( \left( \text{endmant[ch]} - 1 + 9 \right) / 12 \right); \text{forD45} \]

For the LFE channels, the exponents structure in a Dolby Digital frame can be found in [1]. If the LFE channel is on, there are 7 exponents in one audio block. One exponent is the absolute value lfeexps[0] and the other six exponents are combined in two groups in D15 mode.

For the coupling channel, the structure of exponents is similar to FBW channel. There are differences worthy noticing.
First, the first exponent in FBW and LFE channels is a value in use while the first exponent in coupling channel is just a reference value i.e. it is not an actual exponent for a coefficient.

Second, the first value in coupling channel is a 4 bit wide value, but it represents a 5 bit information. Because the LSB is always 0, it is not transmitted. To use this value, the decoder must left shift the 4 bit exponent 1 bit to get an 5 bit exponent.

Decoding a set of coded grouped exponents will create a set of 5-bit absolute exponents [1].

The exponents in a 7 bit group need to be ungrouped. There are three exponents in every group. The decoder need to follow the below equations to ungroup.

\[ M1 = \text{truncate}(\text{gexp}/25) \]  
\[ M2 = ((\text{gexp}\%25)/5) \]  
\[ M3 = (\text{gexp}\%25)\%5 \]

M1, M2 and M3 are called mapped value and gexp is grouped exponent.

To get the differential exponent value dexp, the mapped values need to be subtracted by 2.

To get the actual exponents, the decoder need to add differential value to last exponent \( \text{exp}[n-1] \). [1]

\[ \text{exp}[n] = \text{exp}[n-1] + d\text{exp}[n] \]

For D25 strategy, the \( \text{exp}[n] \) is used as two successive exponents and for D45 strategy, the \( \text{exp}[n] \) is used as four successive exponents.

Pseudo code can be found in reference [1].

### 3.2.3 Bit Allocation

#### Overview

Bit allocation is an very important step in Dolby Digital format. It shows the essential part of Dolby Digital as an audio compression algorithm. The bit allocation utilize the model of human hearing and decide how to distribute how many bits to every coefficient.

Similar to exponents information, the bit allocation information can be shared in one frame, i.e. the range can be one audio block using one set bit allocation information to six audio blocks using one set information. No matter which information sharing case, there is always a new set of bit allocation in the first audio block.

The bit allocation can be divided into sub steps. The algorithm theory behind the steps are not explained in this thesis. The focus is on how to process bit allocation in practice.
Mapping Into PSD and Integration

First, the exponents value need to be mapped to 13 bit signed log power spectral density function.

\[
\text{psd}[\text{bin}] = (3072 - (\text{exp}[\text{bin}] \ll 7))
\]  

(3.10)

The exponents \(\text{exp}[\text{bin}]\) are always 5 bit wide and the range is from 0 to 24 as explained before. So the range of \(\text{psd}[\text{bin}]\) is from 0 to 3072.

For PSD integration, the pseudo code can be found in [1]

This step of the algorithm integrates fine-grain PSD values within each of a multiplicity of 1/6th octave bands. The bndtab[] array gives the first mantissa number in each band. The bndsz[] array provides the width of each band in number of included mantissas. The masktab[], shows the mapping from mantissa number into the associated 1/6 octave band number. These tables contain duplicate information, all of which need not be available in an actual implementation. They are shown here for simplicity of presentation only. The integration of PSD values in each band is performed with log-addition. The log-addition is implemented by computing the difference between the two operands and using the absolute difference divided by 2 as an address into a length 256 lookup table, latab[] [1].

Compute Excitation Function

The excitation function is computed by applying the prototype masking curve selected by the encoder (and transmitted to the decoder) to the integrated PSD spectrum (bndpsd[]). The result of this computation is then offset downward in amplitude by the fgain and sgain parameters, which are also obtained from the bit stream [1]. Pseudo code can be found in the standard document.

Compute Masking Curve

This step computes the masking (noise level threshold) curve from the excitation function, as shown below. The fscod and dbpbcod variables are received by the decoder in the bit stream [1].

Apply Delta Information

The delta bit allocation in Dolby Digital is optional. It can improve the sound quality. The delta bit allocation information can be for both FBW channel and coupling channel, not for LFE channel.

The dba information which modifies the decoder bit allocation are transmitted as side information. The allocation modifications occur in the form of adjustments to the default masking curve computed in the decoder. Adjustments can be made in multiples of \(\pm 6\) dB. On the average, a masking curve adjustment of \(-6\) dB corresponds to an increase of 1 bit of resolution for all the mantissas in the affected 1/6th octave band [1].
3.2 Key Steps

Compute Bit Allocation

The final bit allocation results are calculated in this step.

The sum of all channel mantissa allocations in one syncframe is constrained by the encoder to be less than or equal to the total number of mantissa bits available for that syncframe. The encoder accomplishes this by iterating on the values of csnroffst and fsnroffst (or cplfsnroffst for the coupling or lfefsnroffst for low frequency effects channels) to obtain an appropriate result. The decoder is guaranteed to receive a mantissa allocation which meets the constraints of a fixed transmission bit-rate. At the end of this step, the bap[] array contains a series of 4-bit pointers [1].

3.2.4 Decoding Mantissas

Overview

All mantissas are quantized to a fixed level of precision indicated by the corresponding bap[] [1].

Some quantized mantissa values are grouped together and encoded into a common codeword to further reduce the data amount. In the case of the 3-level quantizer, 3 quantized values are grouped together and represented by a 5-bit codeword in the data stream. In the case of the 5-level quantizer, 3 quantized values are grouped and represented by a 7-bit codeword. For the 11-level quantizer, 2 quantized values are grouped and represented by a 7-bit codeword. [1]

<table>
<thead>
<tr>
<th>bap[]</th>
<th>Quantizer Levels</th>
<th>Quantization Type</th>
<th>Mantissas Bits (group bits/ num in groups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>none</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>symmetric</td>
<td>1.67 (5/3)</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>symmetric</td>
<td>2.33 (7/3)</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>symmetric</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>symmetric</td>
<td>3.5 (7/2)</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>symmetric</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>asymmetric</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>64</td>
<td>asymmetric</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>asymmetric</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>256</td>
<td>asymmetric</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>asymmetric</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>1024</td>
<td>asymmetric</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>2048</td>
<td>asymmetric</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>4096</td>
<td>asymmetric</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>16,384</td>
<td>asymmetric</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>65,536</td>
<td>asymmetric</td>
<td>16</td>
</tr>
</tbody>
</table>

*Table 3.2: Mapping of bap to Quantizer [1]*
Expansion and Ungrouping Mantissas

Expansion of Mantissas can be divided into two cases. For bap[] value from 6 to 15, the quantization type is asymmetric. To decode the mantissa back to fixed point presentation, the decoder just needs to right shift the mantissa by the corresponding exponent.

\[
transform\_coefficient[k] = mantissa[k] >> exponent[k]
\] (3.11)

For bap[] value from 1 to 5, the quantization is symmetrical. The mantissas are in a coded form. The coded should be transformed to two’s complement fractional binary words according to certain lookup table. For example, if bap[] = 2, the lookup table is shown in Table 3.3. Tables for other cases can be found in [1].

<table>
<thead>
<tr>
<th>Mantissa Code</th>
<th>Mantissa Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-4./5</td>
</tr>
<tr>
<td>1</td>
<td>-2./5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2./5</td>
</tr>
<tr>
<td>4</td>
<td>4./5</td>
</tr>
</tbody>
</table>

*Table 3.3: bap[]=2 Mantissa Lookup Table [1]*

Then the mantissa value from lookup tables need to right shifted according to the corresponding exponent to get the transform coefficient [1].

\[
transform\_coefficient[k] = quantization\_table[mantissa[k]] >> exponent[k]
\] (3.12)

For cases that bap[] = 1, 2 and 4, the coded mantissas are grouped together either two or three into a group word. In this way, Dolby Digital can further decrease the data amount. To ungroup the group word, the decoder just needs to follow the following equations.

For bap[]=1,

\[
\begin{align*}
mantissa\_code[a] &= \text{truncate}(\text{group\_code}/9) \\
mantissa\_code[b] &= \text{truncate}((\text{group\_code}\%9)/3) \\
mantissa\_code[c] &= (\text{group\_code}\%9) \%3
\end{align*}
\] (3.13)

For bap[]=2,

\[
\begin{align*}
mantissa\_code[a] &= \text{truncate}(\text{group\_code}/25) \\
mantissa\_code[b] &= \text{truncate}((\text{group\_code}\%25)/5) \\
mantissa\_code[c] &= (\text{group\_code}\%25) \%5
\end{align*}
\] (3.14)

For bap[]=4,

\[
\begin{align*}
mantissa\_code[a] &= \text{truncate}(\text{group\_code}/11) \\
mantissa\_code[b] &= \text{group\_code}\%11
\end{align*}
\] (3.15)
3.2.5 Channel Coupling

The channel coupling technique combines the high frequency part of all the channels into coupling channel. In this way, Dolby Digital format can reduce the data amount without losing much sound quality due to that human ears are not sensitive to high frequency sound.

If enabled, channel coupling is performed on encode by averaging the transform coefficients across channels that are included in the coupling channel. Each coupled channel has a unique set of coupling coordinates which are used to preserve the high frequency envelopes of the original channels. The coupling process is performed above a coupling frequency that is defined by the cplbegf value [1].

In the decoupling process, the information is transformed back to every channel by multiplying the coefficients with the coupling coordinates corresponding to each channel.

Sub Band Structure

In every audio block, there are 256 transform coefficients for each channel. For coupling process, the transform coefficients between the 37th and the 252th are divided into 18 sub bands and therefore each band contains 12 coefficients. The frequency range of each sub-band can be seen in [1].

3.2.6 Inverse Transform

The audio information is carried in form of a number of frequency coefficients. The purpose of inverse transform is to transform frequency coefficients into time domain samples.

In the AC-3 transform block switching procedure, a block length of either 512 or 256 samples (time resolution of 10.7 or 5.3 ms for sampling frequency of 48 kHz) can be employed. Normal blocks are of length 512 samples. When a normal windowed block is transformed, the result is 256 unique frequency domain transform coefficients. Shorter blocks are constructed by taking the usual 512 sample windowed audio segment and splitting it into two segments containing 256 samples each. The first half of an MDCT block is transformed separately but identically to the second half of that block. Each half of the block produces 128 unique non-zero transform coefficients representing frequencies from 0 to \( fs/2 \), for a total of 256. This is identical to the number of coefficients produced by a single 512 sample block, but with two times improved temporal resolution. Transform coefficients from the two half-blocks are interleaved together on a coefficient-by-coefficient basis to form a single block of 256 values. This block is quantized and transmitted identically to a single long block [1].
4 Hardware Platform

4.1 Introduction

The company’s current product has a FPGA inside, which is used for audio signal processing. There is some resource left on the FPGA which gives the possibility to integrate an AC3 decoding unit into the current system.

4.2 Xilinx SP601 Evaluation Board

A diagram of the board can be found in Figure 4.1 [16].

This board has a Spartan-6 XC6SLX16 FPGA as the main component. This FPGA chip has 2278 logic slices and 32 DSP48A1 slices. Every Spartan-6 FPGA slice has four LUTs and eight flip-flops. Every DSP48A1 slice has an 18 x 18 multiplier, an adder, and an accumulator. Also it has 32 18Kb block RAM blocks. These resource are relevant to this thesis.

A Xilinx SP601 board is provided for this thesis.

4.3 Xilinx ISE

Integrated Software Environment (ISE) is a program for synthesis and analysis of hardware description language designs on Xilinx FPGA. This program compiles the code and synthesizes the design. It generates a bit file that can be loaded into an FPGA.
Figure 4.1: Spartan-6 FPGA SP601 Evaluation Board [15]
4.4 PicoBlaze Micro Controller

PicoBlaze micro controller is an 8-bit soft processor which can be synthesized on Xilinx Spartan-6 FPGA and it is free.

4.4.1 Introduction

PicoBlaze is also called KCPSM, which is in short of Constant(K) Coded Programmable State Machine. It has multiple versions. The version used in this thesis is KCPSM6. Probably its greatest strengths are that it is 100% embedded and requires only 26 logic Slices and a Block Memory which equates to 4.3% of the smallest XC6SLX4 and just 0.11% of the XC6SLX150T [14].

In AC3 audio decoding process, there are many steps such as breaking the bit stream into blocks, breaking blocks into different data, and controlling the order of different steps. These steps are sequential.

In simple terms, hardware is parallel and processors are sequential. So converting a small amount of hardware into a processor is often a more efficient way to implement sequential functions such as state machines (especially complex ones) or to time-share hardware resources when there are several slower tasks to be performed. It is also more natural to describe sequential tasks in software whereas HDL is best at describing parallel hardware [14].

4.4.2 Architecture

PicoBlaze can execute maximum 4K instruction and it takes two clock cycles to execute one instruction. In Spartan-6 FPGA, it can be stable working at 105MHz, which gives up to 52 MIPS performance.

PicoBlaze has 16 general purpose registers. All operations can be performed using any register (i.e. there are no special purpose registers) so you have complete freedom to allocate and use registers when writing your programs[14].

PicoBlaze provides 49 instructions including ALU operations like AND, OR, ADD, SUB instructions, Compare instructions, Shift and Rotate instructions, Input and Output, Jump and Call instructions, etc.

PicoBlaze can be reset, supports one mask-able interrupt with acknowledge, and a ‘sleep’ control can be used to suspend program execution for any period of time to save power, wait under hardware control (e.g. Handshaking) or to enforce slower execution rate relative to the clock frequency[14].

4.4.3 Components and Connections

The components of the design can be found in Figure 4.2.

From Figure 4.2, ‘kcpsm6’ is the processor core and ‘your_program’ is the memory which stores your code. Due to the address signals are 12 bits width, PicoBlaze can support code with maximum 4K instructions.

Programs for PicoBlaze are stored in Block Memory in FPGAs. The size of the memory is configurable according to the size of the program or the device.
4.4.4 Program Environment

The program for PicoBlaze needs to be written as a standard text file and then saved with the `.psm` file extension.

The KCPSM6 assembler reads and assembles your program (PSM file) into the instruction codes. It then reads an HDL template file called ‘ROM_form.vhd’ (or ROM_form.v) into which it inserts your assembled program and writes out the HDL file defining the program memory containing the program for use in the design[14]. It is shown as in Figure 4.3.

The assembler generates the ‘your_program.vhd’ (or .v) definition of the pro-
gram memory ready to include in the ISE project.

Each time the PSM file is modified, the assembler needs to run again so that the changes are also included in the HDL program memory definition file. Detailed steps to programming PicoBlaze can be found in [14].
5.1 Propose

The AC3 frame has a tricky structure. The former information bits decide if the following information bits exist or not. In this case, if pure VHDL logic blocks are used to parse the frame, it will cost a lot logic resources. Due to this reason, PicoBlaze – an integrated microcontroller is used to parse the frame into data, side information, etc. and control other hardware modules in order to control the whole decoding flow.

Therefore, the following structure is proposed, shown in Figure 5.1.

Hardware blocks are divided as shown in Figure 5.1, which is according to the functionality of each block as described in the standard. They are not included in the PicoBlaze because they contain data processing more than 8 bit width and operations which are not supported by PicoBlaze instructions, for example division operation.

5.2 Data Flow

In the system, first, the bit stream is sent to a FIFO. The microcontroller PicoBlaze data from the FIFO and parses the audio blocks one by one.

After the bit allocation information in one audio block is parsed. The PicoBlaze will pause and enable exponent decoder to decoder packed exponents to unpacked exponents and stores them in to unpacked exponent block ram. Bit allocation block calculate the bap information using exponent. The bap information decides how many bits are distributed to each mantissa. PicoBlaze uses the bap information to continue parsing the rest of the audio block and stores the grouped mantissas into block ram. The mantissa decoder block generates the transform coefficients by using exponents and mantissas. IMDCT transfers the
information from frequency domain to time domain and store the PCM data in the output FIFO. During the whole process, PicoBlaze controls the data flow and every block. PicoBlaze is idle during the hardware block operation and hardware blocks are relatively fast comparing to PicoBlaze data parsing.

5.3 Implementation

5.3.1 PicoBlaze

PicoBlaze is a small 8 bit microcontroller and it can be instanced many times in a FPGA design. It costs little resources which is only 26 logic Slices in Spartan-6 and a Block Memory. And it is fully embedded. To embed it to the design, it is only required basic hardware description code for connections.

Due to the input port of PicoBlaze is 8 bit wide and the fact that the former information decides if the following information exists or not, 8 bits data is read into a register in PicoBlaze every time and shift out information bit one by one to analyze. The whole program for the microcontroller is written in assembly code. And the input and output network are built in VHDL.
PicoBlaze Assembly

The PicoBlaze does not have a C compiler but it comes with an assembler. So the decoding program is written in PicoBlaze assembly. The program is divided into three main parts, shown in following program.

Listing 5.1: decoding.psm

;===================================
; main program
;===================================

forever:
call synchr
call bsi
call audioblock
jump forever

The syncword is 0x0B77 which is the most significant bit transmitted first. The synchr subroutine is to detect synchronization, CRC value, sample rate code and frame size of this frame. The CRC word is a 16-bit word which can be used to check the first 5/8 of the syncframe. Sample rate code is to indicate which sample rate is used, for example, '00' means 48kHz. Frame size is to determine how many words inside this frame where every word is 16 bits.

The bsi subroutine is used to decode the Bit Stream Information block. This block contains information of coding mode which is telling information like if surround channels are in use or if two completely independent channels are in use. It also contains minor information such as room type, language code, copyright etc. which is not relevant to the main functionality. Details can be found in [1].

The audioblock subroutine is the main part of the assembly code. It has a structure like following code.

Listing 5.2: decoding.psm

;===================================
; audioblock program
;===================================

audioblock:
load sD, 06

label_audioblock:
call exponent
call bit_allocation
call mantissa
sub sD, 01
jump nz, label_audioblock
return
The exponent subroutine is to extract exports data out of the audio block along with other auxiliary information like coupling begin frequency code, coupling end frequency code, coupling band structure information. The extracted exports data are stored in a BRAM and to be utilized.

The bit_allocation subroutine is to extract information data for bit allocation calculation. Information are like slow decay code, coarse SNR offset, coupling fast leak initialization and etc. These information are outputs from PicoBlaze output network and input to bit allocation calculation block for calculation which get bit allocation information from exponents. Details can be found in [1].

The mantissa subroutine is to extract mantissa data out of the audio block. How many bits in every group mantissa are based on the bap information. PicoBlaze stores the grouped mantissas into block ram. After mantissa being extracted from one audio block, mantissa decoder will process the mantissas and exponents into transform coefficients. And the transform coefficients are stored in a BRAM too.

PicoBlaze out_port is 8 bits wide and port_id is 8 bits wide, so PicoBlaze can output 8-bit values to up to 256 general purpose output ports using its OUTPUT instructions. When PicoBlaze executes an OUTPUT instruction it sets port_id to specify which of 256 ports it wants to write the 8-bit data value present on out_port. A single clock cycle enable pulse is generated on write_strobe and hardware uses write_strobe to qualify the decodes of port_id to ensure that only the correct register captures the out_port value. [14]

5.3.2 Exponent Decoder

The audio data is a number of frequency coefficients. And each coefficient is represented by an exponent and a mantissa. Each exponent has 5 bits, but the maximum value is 24. All audio data is not shared between frames. Exponents can be shared between blocks within one frame. So after we extract exponents from the first audio block from a frame, they should be stored in a BRAM for further usage.

To make the coding more efficient, The first exponent of every channel is a 4 bit value and it is used as a reference for following exponents. Also the following exponents are packed further. It has 3 exponent strategies, which are D15, D25 and D45, details can be found in Chapter 3.

Pseudo Code

The function of exponent decoder is to use all side information to decode packed exponents to independent unpacked exponents.

The decoding pseudo code is like following listing.
Listing 5.3: Exponent Decoding [1]

/* unpack the mapped values */
for (grp = 0; grp < ngrps; grp++)
|
    expacc = gexp[grp] ;
    dexp[grp * 3] = truncate (expacc / 25) ;
    expacc = expacc - (25 * dexp[grp * 3]) ;
    dexp[(grp * 3) + 1] = truncate (expacc / 5) ;
    expacc = expacc - (5 * dexp[(grp * 3) + 1]) ;
    dexp[(grp * 3) + 2] = expacc ;
|
/* unbiased mapped values */
for (grp = 0; grp < (ngrps * 3); grp++)
|
    dexp[grp] = dexp[grp] - 2 ;
|
/* convert from differentials to absolutes */
prevexp = absexp ;
for (i = 0; i < (ngrps * 3); i++)
|
    aexp[i] = prevexp + dexp[i] ;
    prevexp = aexp[i] ;
|
/* expand to full absolute exponent array, using grpsize */
exp[0] = absexp ;
for (i = 0; i < (ngrps * 3); i++)
|
    for (j = 0; j < grpsize; j++)
    |
        exp[(i * grpsize) + j +1] = aexp[i] ;
    |
|
Where:
ngrps = number of grouped exponents
grpsize = 1 for D15
= 2 for D25
= 4 for D45
absexp = absolute exponent (exps[ch][0], (cplabsexp«1), or lfeexps[0])

Hardware Implementation

The exponent decoder is connected to BRAMs which store packed exponents and unpacked exponents. During the decoding process, the module will read out packed exponents from BRAMs, decode according to the pseudo code, writes the results which are the unpacked exponents into the corresponding BRAMs.
Inputs `din_fbw` are grouped exponents data for fbw channel and `din_cpl` are grouped exponents data for cpl channel. Inputs `exps_chn_0` are the first exponent for every fbw channel. Inputs `nchgrps` are the number of group exponents in every channel. Inputs `lfeexps` are group exponents in lfe channel. `cplinu` is signal showing coupling is used or not. Output `dout` are ungrouped exponents for every channel to be stored in corresponding BRAM.

For the exponent decoder block, an FSM is implemented to control the data flow. The exponents of coupling channel, all FSW channels and LFE channel are decoded in different states. When all exponents are done, exponent decoder sends a enable signal to bit allocation module to continue the process.
Figure 5.3: Exponents Decoder FSM
5.3.3 Bit Allocation

Bit allocation is divided into 6 steps. The results of every step are stored in block RAM for the following calculation steps. The diagram is shown in Figure 5.4.

Figure 5.4: Bit Allocation Diagram
Initialization and Exponent Mapping into PSD

Initialization step is to calculate parameters for bit allocation from the information in the data stream. For example, the start and end frequencies can be simply computed from the following pseudo code. More information can be found in [1].

**Listing 5.4: Initialization [1]**

```c
/* for fbw channels */
for (ch=0; ch<nfchans; ch++)
{
    strtmant[ch] = 0;
    /* channel is coupled */
    if (chincpl[ch]) endmant[ch] = 37 + (12 * cplbegf) ;
    /* channel is not coupled */
    else endmant[ch] = 37 + (3 * (chbwcod + 12)) ;
}
/* for coupling channel */
cplstrtmant = 37 + (12 * cplbegf) ;
cplendmant = 37 + [12 * (cplendf + 3)] ;
/* for lfe channel */
lfestarment = 0 ;
lfeendmant = 7 ;
```

Exponent Mapping step is to decode exponents into power spectral density function, which requires simply shift operation.

**Listing 5.5: Exponent Mapping into PSD [1]**

```c
for (bin=start; bin<end; bin++)
{
    psd[bin] = (3072 - (exp[bin] << 7)) ;
}
```
Input din_exponents are the ungrouped exponents, start_bin and end_bin are calculated during PicoBlaze data parsing.

**PSD Integration**

PSD integration step is to integrate PSD data. The bndtab[] array gives the first mantissa number in each band. The bndsz[] array provides the width of each band in number of included mantissas. These two tables contain duplicate information, all of which need not be available in an actual implementation. They are shown here for simplicity of presentation only. The integration of PSD values in each band is performed with log-addition. The log-addition is implemented by computing the difference between the two operands and using the absolute difference divided by 2 as an address into a length 256 lookup table, latab[]. [1] These tables can be found in the standard document.
Listing 5.6: PSD Integration [1]

j = start ;
k = masktab[start] ;
do
 |
  lastbin = min(bndtab[k] + bndsz[k], end);
bndpsd[k] = psd[j] ;
j++ ;
for (i = j; i < lastbin; i++)
 |
  bndpsd[k] = logadd(bndpsd[k], psd[j]) ;
j++ ;
} k++ ;
while (end > lastbin) ;
logadd(a, b)
 |
  c = a - b ;
  address = min((abs(c) >> 1), 255) ;
  if (c >= 0)
  |
    return(a + latab(address)) ;
  }
else
 |
  return(b + latab(address)) ;
}
Input din_psd are the output from the PSD which is stored in a BRAM. Output bndpsd is stored into a BRAM for next step.

**Compute Excitation Function**

The excitation function is computed by applying the prototype masking curve selected by the encoder (and transmitted to the decoder) to the integrated PSD spectrum (bndpsd[]). The result of this computation is then offset downward in amplitude by the fgain and sgain parameters, which are also obtained from the bit stream. [1] Pseudo Code can be found in [1].
5.3 Implementation

Input dbpbcod, fscod, sgain and fgain are calculated during PicoBlaze data parsing. Their meaning can be found in [1].

**Compute Masking Curve**

Compute Masking Curve is to compute the noise level threshold curve. It is to use fscod and dbpbcod parameters in the data stream to search in hth[][] hearing threshold table.

**Listing 5.7: Compute Masking Curve [1]**

```c
for (bin = bndstrt; bin < bndend; bin++)
{
    if (bndpsd[bin] < dbknee)
    {
        excite[bin] += ((dbknee - bndpsd[bin]) >> 2);  
    }
    mask[bin] = max(excite[bin], hth[fscod][bin]);
}
```
Input dbpbcod, fscod, sgain and fgain are calculated during PicoBlaze data parsing. din_bndpsd and din_excite are output from previous steps.

**Apply Delta Bit Allocation**

The optional delta bit allocation information in the bit stream provides a means for the encoder to transmit side information to the decoder which directly increases or decreases the masking curve obtained by the parametric routine. Delta bit allocation can be enabled by the encoder for audio blocks which derive an improvement in audio quality when the default bit allocation is appropriately modified. The delta bit allocation option is available for each fbw channel and the coupling channel. [1]
5.3 Implementation

Listing 5.8: Apply Delta Bit Allocation [1]

if ((deltbae == 0) || (deltbae == 1))
{
    band = 0;
    for (seg = 0; seg < deltnseg+1; seg++)
    {
        band += deltoffst[seg];
        if (deltba[seg] >= 4)
        {
            delta = (deltba[seg] - 3) << 7;
        }
        else
        {
            delta = (deltba[seg] - 4) << 7;
        }
        for (k = 0; k < deltlen[seg]; k++)
        {
            mask[band] += delta;
            band++;
        }
    }
}
Input deltnseg, deltoffst, delt etc. are calculated during PicoBlaze data parsing. Their meaning can be found in [1].

**Compute Bit Allocation**

Compute Bit Allocation is the final step in the bit allocation process, which will generates bap[] array. Bap[] array is used by PicoBlaze to further parse the data stream.
Listing 5.9: Apply Delta Bit Allocation [1]

\[
i = \text{start} ; \\
j = \text{masktab[start]} ; \\
\text{do} \\
\{ \\
\hspace{1em} \text{lastbin} = \min(\text{bndtab}[j] + \text{bndsz}[j], \text{end}) ; \\
\hspace{1em} \text{mask}[j] -= \text{snroffset} ; \\
\hspace{1em} \text{mask}[j] -= \text{floor} ; \\
\hspace{1em} \text{if} (\text{mask}[j] < 0) \\
\hspace{2.5em} \{ \\
\hspace{4em} \text{mask}[j] = 0 ; \\
\hspace{2.5em}\} \\
\hspace{1em} \text{mask}[j] &= 0x1fe0 ; \\
\hspace{1em} \text{mask}[j] += \text{floor} ; \\
\hspace{1em} \text{for} (k = i ; k < \text{lastbin} ; k++) \\
\hspace{2.5em} \{ \\
\hspace{4em} \text{address} = (\text{psd}[i] - \text{mask}[j]) >> 5 ; \\
\hspace{4em} \text{address} = \min(63, \max(0, \text{address})) ; \\
\hspace{4em} \text{bap}[i] = \text{baptab[address]} ; \\
\hspace{4em} i++ ; \\
\hspace{2.5em}\} \\
\hspace{1em} j++; \\
\} \\
\text{while} (\text{end} > \text{lastbin}) ;
\]
5.3.4 Mantissa Ungroup and Mantissa Decoder

For Mantissa Ungroup step, it needs to follow the following rules. For cases that \( bap[] = 1, 2 \) and 4, the coded mantissas are grouped together either two or three into a group word. In this way, Dolby Digital can further decrease the data amount. To ungroup the group word, the decoder just needs to follow the following equations.

For \( bap[]=1 \),

\[
\begin{align*}
\text{mantissa}_a & = \text{truncate}(\text{group}_c/9) \\
\text{mantissa}_b & = \text{truncate}((\text{group}_c\%9)/3) \\
\text{mantissa}_c & = (\text{group}_c\%9)\%3
\end{align*}
\] (5.1)

For \( bap[]=2 \),
5.3 Implementation

\[
\text{mantissa\_code}[a] = \text{truncate}(\text{group\_code}/25)
\]
\[
\text{mantissa\_code}[b] = \text{truncate}((\text{group\_code}\%25)/5)
\]
\[
\text{mantissa\_code}[c] = (\text{group\_code}\%25)\%5
\]

For \text{bap}[] = 4,

\[
\text{mantissa\_code}[a] = \text{truncate}(\text{group\_code}/11)
\]
\[
\text{mantissa\_code}[b] = \text{group\_code}\%11
\]

For Mantissa Decoder, it basically right shift mantissa by corresponding exponent to recover the fixed point form. It is divided into two cases. For \text{bap}[] value from 1 to 5, the quantization is symmetrical. The mantissas are in a coded form. The coded should be transformed to two's complement fractional binary words according to certain lookup table. Then the mantissa value from lookup tables need to right shifted according to the corresponding exponent to get the transform coefficient [1].

\[
\text{transform\_coefficient}[k] = \text{quantization\_table}[\text{mantissa}[k]] \gg \text{exponent}[k]
\]

For \text{bap}[] value from 6 to 15, the formula changed to:

\[
\text{transform\_coefficient}[k] = \text{mantissa}[k] \gg \text{exponent}[k]
\]

**Hardware Implementation**

According to the \text{bap} information, the ungrouping of mantissa can be done by dividing the mantissa in the same divider in exponent decoder. The function of this block is to arithmetic right shift the mantissa by corresponding exponent.

**5.3.5 IMDCT**

The FPGA implementation diagram is based on the fast algorithm in the AC3 standard document. The process for 512-sample IMDCT transform is explained here.
1. The 256 coefficients \(X[k]\) are ready.
2. Apply Pre-IFFT complex multiply.

**Listing 5.10: Pre-IFFT complex multiply [1]**

```
for (k=0; k<N/4; k++)
|
    Z[k] = (X[N/2-2*k-1]*xcos1[k]-X[2*k]*xsin1[k])
    + j*(X[2*k]*xcos1[k]+X[N/2-2*k-1]*xsin1[k]);
```

where

\[
\text{xcos1}[k] = -\cos (2\pi\cdot (8\cdot k + 1)/(8\cdot N));
\]
\[
\text{xsin1}[k] = -\sin (2\pi\cdot (8\cdot k + 1)/(8\cdot N)).
\]
3. Apply 128 point complex IFFT.

**Listing 5.11: 128 point IFFT [1]**

```c
for (n=0; n<N/4; n++)
{
    z[n]=0;
    for (k=0; k<N/4; k++)
    {
        z[n] += Z[k] * (cos(8*pi*k*n/N) + j*sin(8*pi*k*n/N));
    }
}
```

4. Post-IFFT complex multiply.

**Listing 5.12: Pre-IFFT complex multiply [1]**

```c
for (n=0; n<N/4; n++)
{
    y[n] = (zr[n]*xcos1[n] - zi[n]*xsin1[n])
    + j*(zi[n]*xcos1[n] + zr[n]*xsin1[n]);
}
```

where

```c
zr[n] = real(z[n]);
z[i][n] = imag(z[n]);
```

and xcos1[n] and xsin1[n] are as defined in step 2 above.

5. Compute windowed time domain samples.

**Listing 5.13: Windowing [1]**

```c
for (n=0; n<N/8; n++)
{
    x[2*n] = -yi[N/8+n] * w[2*n];
x[2*n+1] = yr[N/8-n-1] * w[2*n+1];
x[N/4+2*n] = -yr[n] * w[N/4+2*n];
x[N/4+2*n+1] = yi[N/4-n-1] * w[N/4+2*n+1];
x[N/2+2*n] = -yr[N/8+n] * w[N/2-2*n-1];
x[N/2+2*n+1] = yi[N/8-n-1] * w[N/2-2*n-2];
x[3*N/4+2*n] = yi[n] * w[N/4-2*n-1];
x[3*N/4+2*n+1] = -yr[N/4-n-1] * w[N/4-2*n-2];
}
```

where

```c
yr[n] = real(y[n]);
yi[n] = imag(y[n]);
```

w[n] is the transform window sequence which can be found in [1].

6. Overlap and add.

**Listing 5.14: Overlap and add [1]**
for \( n=0; n<N/2; n++ \)
\[
pcm[n] = 2 \times (x[n] + delay[n]) ; \\
delay[n] = x[N/2+n] ;
\]

In the FPGA implementation, the complex multipliers, IFFT blocks and block RAMs are generated by Core Generator in Xilinx ISE.

The hardware structure of IMDCT is shown in Figure 5.11.

This is a 512 sample IMDCT transform. All Real and Imag BRAMs sizes are 18bit\(^{\times}128\). The size of BRAM after Window function is 18bit\(^{\times}512\). The size of Delay BRAM is 18bit\(^{\times}256\).

The complex multiplier blocks consist of three main parts, one complex multiplier and two ROMs. The multiplier is generated by the core generator in Xilinx ISE. Channel A and B are both 18 bits wide. It uses hardware multiplier in the chip and has nonblocking flow control. To use the least amount of resources, the optimization goal is set to ‘Resource’. The two ROMs are also generated by the core generator. They contains the xcos1[k] and xsin1[k], which are used in step 2 and 4. The values of xcos1[k] and xsin1[k] are calculated by using Matlab. IFFT block is generated by the core generator in Xilinx ISE. Due to PicoBlaze can run at maximum around 100MHz on Spartan-6, the target clock frequency is set to 100MHz and Implementation option is set to ‘Radix-2, Burst I/O’ to save area. The Phase Factors are set to store in BRAM. The 3 multiplier structure is selected to achieve resource optimization.
Figure 5.11: IMDCT diagram
6

Results and Conclusion

6.1 Results

The following synthesis results are from Xilinx ISE Design Suite 13.4. The Optimization Goal is set to ‘Area’ to minimize the area usage. The Optimization Effort is set to ‘High’.

<table>
<thead>
<tr>
<th>Component</th>
<th>Slice Registers</th>
<th>Slice LUTs</th>
<th>Block RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicoBlaze with input/output network</td>
<td>142</td>
<td>288</td>
<td>2</td>
</tr>
<tr>
<td>Exponent Decoder</td>
<td>155</td>
<td>662</td>
<td>0</td>
</tr>
<tr>
<td>PSD</td>
<td>18</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>PSD_integration</td>
<td>127</td>
<td>132</td>
<td>2</td>
</tr>
<tr>
<td>Compute_Excitation_Function</td>
<td>125</td>
<td>192</td>
<td>1</td>
</tr>
<tr>
<td>Compute_Masking_Curve</td>
<td>33</td>
<td>93</td>
<td>2</td>
</tr>
<tr>
<td>Delta_Bit_Allocation</td>
<td>48</td>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>Compute_Bit_Allocation</td>
<td>70</td>
<td>144</td>
<td>2</td>
</tr>
<tr>
<td>Mantissa_Ungroup</td>
<td>36</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>Mantissa_Decoder</td>
<td>68</td>
<td>134</td>
<td>0</td>
</tr>
<tr>
<td>IMDCT</td>
<td>1345</td>
<td>1180</td>
<td>6</td>
</tr>
</tbody>
</table>

*Table 6.1: Synthesis Results*

The goal clock frequency is 100MHz, due to the limitation of PicoBlaze on Spartan6 FPGA. The synthesis result shows that the maximum frequency of the design is 110.142MHz.
6.2 Conclusion

From the synthesis result, it shows that the proposed solution takes reasonable hardware resources. So the current implementation is able to fit into a tiny FPGA chip. This is not including the functions which are not considered for this project. Also the integrating to the current FPGA in the product can take extra resource.

In summary of all the work mentioned above in Chapter 1 and this thesis, it is a widely accepted opinion that the AC3 decoding is better to be done with a hardware/software co-design solution.

Also pseudo codes in AC3 standard are very similar to C. So a MCU with floating point calculation will be very suitable for such task.

6.3 Future Work

- Implementation can be optimized. The current design may not be the best solution. The area/resource usage can be further decreased. For example, some BRAM can be reused during different steps. IMDCT module is currently based on Xilinx core generator, but it may take less area if it is dedicately designed.

- In the current implementation, PicoBlaze is stalled during hardware module operation, which is shown in Figure 6.1. There is possibility to put more hardware function into PicoBlaze, for example, Exponent Decoding, which makes the timing more compact.

![Decoding Timing Diagram](image)

**Figure 6.1: Decoding Timing Diagram**

- The system needs further verification. Verification on module level has been done on this thesis. But the system level verification is not done very well. It is partly because of limited time and also because there is no really efficient and good method to verify.

- Dynamic range compression and down-mixing are optional. They are not included into this project. But it can improved the sound quality. So some further work can be done on that.
• According to online information, there are already open source C code projects for AC3 decoding. So it is worth to explore the possibility to use a MCU to decode AC3 data stream, which may be more suitable.


