Real-time Automatic Transcription of Drums Music Tracks on an FPGA Platform

Georgios Papanikas
s090845

Kongens Lyngby Spring 2012

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Στην Εύη και το Θανάση

Dedicated to Evi and Thanasis

Preface

This thesis was prepared at Informatics Mathematical Modelling, the Technical University of Denmark in partial fulfillment of the requirements for acquiring the degree of Master of Science in engineering.

The thesis deals with the automatic transcription of drums music. Literature survey, mainly focused on limited computational complexity of the algorithms, has been conducted. Based on simulation trials' results, an FPGA system is designed and implemented, which recognizes in real-time the instruments of a limited drum kit through the use of a single microphone.

Lyngby, April 2012
Georgios Papanikas

Acknowledgements

I would like to thank my supervisor, Alberto Nannarelli, for his support and attitude throughout the project and, most of all, for the freedom he gave me on the topic's selection and the system's implementation.

Special thanks should be addressed to Mikkel Nørgaard Schmidt for the discussions on the available approaches to NNMF based methodologies. Also, to Per Friis for his quick responses on the licensing problems of some tools, and Sahar Abbaspour for borrowing the development board whenever I needed it.

The contribution of Nikos Parastatidis and Giannis Chionidis on the recordings and various technical issues was invaluable. As it was the feedback on the text from Mara Vasileiou.

Contents

Preface ...iii

Acknowledgements ...v

1. Introduction 1
   - 1.1. Automatic music transcription ...1
   - 1.2. Drums' transcription and sound characteristics ...2
   - 1.3. Thesis' objectives and structure ...5

2. Background and related work 7
   - 2.1. Approaches to drums transcription ...7
     - 2.1.1. Pattern recognition approaches ...7
     - 2.1.2. Separation-based approaches ...9
       - 2.1.2.1. Sources and components ...9
       - 2.1.2.2. Input data representation ...11
       - 2.1.2.3. The approximated factorisation ...12
       - 2.1.2.4. Klapuri's audio onset detection algorithm ...14
   - 2.2. Non-Negative Matrix Factorisation based approaches ...14
     - 2.2.1. Update rules and cost functions ...15
     - 2.2.2. Using prior knowledge about the sources ...16
     - 2.2.3. Relevant work on NNMF-based transcription systems ...18
   - 2.3. The Fourier transform ...20
     - 2.3.1. Window functions ...21
     - 2.3.2. The Short-time Fourier Transform (STFT) ...24

3. Implemented transcription algorithm and simulation 27
   - 3.1. Introduction ...27
   - 3.2. Recordings ...27
     - 3.2.1. Recording equipment ...28
     - 3.2.2. Tempo, note's value and time signature in the common musical notation scheme ...29
     - 3.2.3. Test and training samples ...31
   - 3.3. Algorithm's pseudocode ...32
   - 3.4. Simulation results ...34
     - 3.4.1. Determining frame's overlapping level and length ...34
     - 3.4.2. Determining the window function ...37
     - 3.4.3. Determining the frequency bands ...37
     - 3.4.4. Determining the convergence threshold ...39

3.4.5. Determining the number of components per source 40
3.4.6. Testing seven-instruments rhythms 42

4. Hardware's design and implementation 43
4.1. System's overview 43
4.2. WM8731 audio CODEC 44
4.2.1. Initialization of WM8731 46
4.2.2. Fetching the ADC samples 48
4.3. Window function 50
4.4. Discrete Fourier Transform 52
4.5. Bandwise magnitude sums 54
4.6. Non-Negative Matrix Factorization 56
4.6.1. Steps 1-3 57
4.6.2. Steps 4-5 60
4.7. Latency and onset time detection 61

5. Conclusion 63

References 65

Appendix A 67

Appendix B 69

Appendix C 71

Appendix D 75

I

Introduction

1.1 Automatic music trancription

Automatic music trancription is defined as the "analysis of an acoustic music signal so as to write down the pitch, onset time, duration and source of each sound that occurs in it" [1]. Usually symbols of notes are used in order to indicate these parameters, but depending on the type of music and the instruments taking part in it, written music may take various forms. Applications of automatic music trancription comprise:

- Music information retrieval (MIR) applications. MIR is an interdisciplinary field (combines knowledge from musicology, psycho-acoustics, signal processing, machine learning, etc) about the extraction of musically meaningful information from live or recorded music. Examples of MIR applications are recommender systems (which suggest new songs based on their similarity to a user's input song), genre/artist identification, music generation (combination of the information retrieved from a track with mathematical models in order to generate algorithmic music) and source separation (which decomposes multi-instrument music into tracks containing only one instrument or type of sound).
- (Real time) music processing, such as changing the loudness, the pitch, or the timings of specific sound events.
- Human-computer interaction in various applications, such as musically oriented computer games or instructive applications (electronic tutors).
- Music related equipment, such as music-synchronous light effects.

A complete automatic transcription containing the pitch, the timing information and the source instrument of every single sound event is hardly achievable in "real-world"

music. In many cases the goal is redefined as being able to transcribe only some well-defined part of the music signal, such as the dominant melody or the most prominent drum sounds (partial transcription). What is achievable could be intuitively estimated by considering what an average listener perceives while listening to music. Although recognizing musical instruments, tapping along the rhythm, or humming the main melody are relatively easy tasks, this is not the case with more complex aspects like recognizing different pitch intervals and timing relationships.

The motivation regarding the real-time automatic transcription came into life after the flourish of such systems in industry, during the last years. More and more interactive applications (such as electronic tutor systems or games) which require the real-time processing and feedback on musically meaningful sound events are being developed[1]. In most of the cases they are based on digital interfaces, such as MIDI, in order to recognise these events, but this limits the instruments that could be used to some of the electronic ones. In other cases the information about the sound events is indirectly extracted. For example, through the use of piezoelectric sensors attached to the skins of the drums, someone could get the information regarding when a specific instrument was hit, without dealing at all with the sound itself. In the general case, though, a drummer (still) uses acoustic drum kits and the most straightforward recognition approach is based on the use of a single microphone's input, rather than the use of several microphones setups or sensors which indirectly provide some aspects of the transcription.

1.2 Drums' transcription and sound characteristics

This project's topic is the real-time automatic transcription of a polyphonic music signal which consists of sounds originated from the various instruments of a typical drum kit (illustrated in figure 1.1). It is assumed that other instruments are absent. "Polyphonic" means that more than one notes (or better *strokes* in case of percussions) may exist simultaneously on two, or more, different instruments of the drum kit. The absence of vocals, other instruments' sounds and of any loud noise in general, in combination with the intrinsic characteristics of the drums' sounds, simplify our task. According to the definition of the automatic music transcription four different aspects of a music signal's sound events have to be recognized:

- the duration
- the pitch
- the onset time
- the source instrument

However, in the drums-only case both duration and pitch are not critical, or even meaningful in the typical percussion instruments of a drum kit.

1 Few examples are the Guitar Hero, the Rocksmith and the JamGuru.

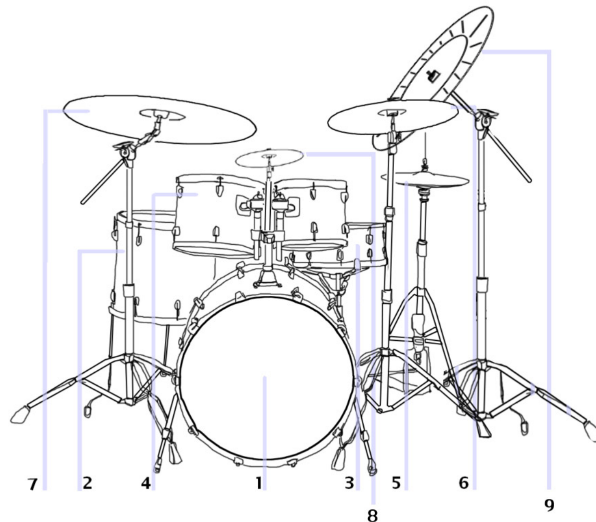


Figure 1.1[2]: 1. Bass drum, 2. floor tom, 3. snare drum, 4. hanging toms, 5. hi-hat cymbal, 6. crash cymbal, 7. ride cymbal, 8. splash cymbal, 9. china type cymbal

If the transcription's target was music played by an instrument like violin or saxophone, finding out the starting time of a note would not be enough; it would be also necessary to know how much time the sound lasted. This duration is not fixed, but determined by the violin/saxophone player, and as such it must be found as part of the transcription procedure. In the case of drums-only music the duration of a sound event is not of our interest, since strokes on a specific drum/cymbal produce sounds of more or less the same duration; and that duration is very short.

A worth noting difference is between *membranophones* (instruments with a skin/membrane stretched across a frame – snare drums, bass drums, tom-toms, etc.) and *idiophones* (the metal plates – ride, crash, hi-hat, etc.): membranophones' sounds are shorter than idiophones' ones. A stroke on a typical bass drum could last 100-200ms, while a stroke on a ride cymbal five to ten times more. In both cases, though, it takes only few milliseconds for the signal's energy to reach its peak and begin to decay. The top of figure 1.2 illustrates the time-domain signals of a stroke on a bass drum (left) and on a ride cymbal (right). The bottom part depicts their spectrograms. The sound of the bass stroke is inaudible after 100-200ms, while after the same period ride's sound is still audible but considerably limited to a small subset of its initial frequency content.

Beyond the duration, the pitch is also not of our interest. Pitch is a property closely related, but not equivalent, to frequency. It allows the ordering of the perceived sounds on a frequency-related scale. The instruments of a typical drum kit are considered unpitched[3], meaning that they are normally not used to play melodies. Their sounds

2 The figure is taken from http://en.wikipedia.org/wiki/Drum_kit

3 There are pitched percussion instruments, not found in drum kits, though. Some characteristic ones are the balafon, the marimba, the xylophone and the tabla.

contain such complex harmonic overtones and wide range of prominent frequencies that no pitch is discernible. The sounds of the different type of strokes on a specific instrument (meaning the different intensities of the hit, hitting the drum/cymbal with a different type of stick or hitting it on a different spot) contain frequencies in the same, more or less, wide ranges. Membranophones tend to have most of their spectral energy in the lower range of the frequency spectrum, typically below 1000Hz, while idiophones' spectral energy is more evenly spread out, resulting in more high-frequency content [2]. This is clearly shown in figure 1.2. Below the lower membrane of the snare drum (the one not being hit by the stick) there is a metal belt attached, whose vibrations cause the existence of more high-frequency energy in snare drum's sounds.

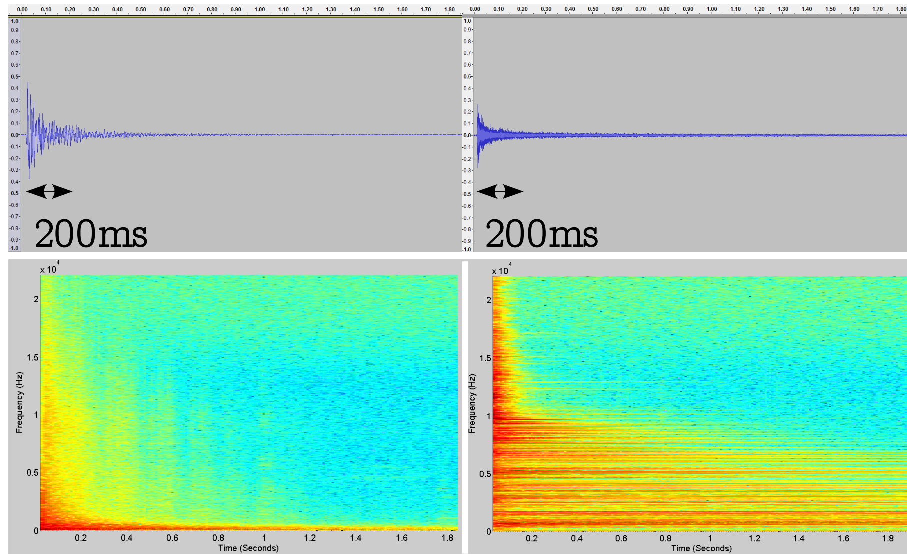


Figure 1.2: Bass and ride strokes signals in time-domain (top) and in frequency-domain (bottom)

It is common in practice to describe a sound's temporal characteristics with the attack time, the decay time, the sustain level and the release time (ADSR – figure 1.3). The attack time refers to the initial phase of the sound event where the amplitude of the signal begins from zero and reaches an (initial) peak. The sustain phase is absent in drums' sounds.

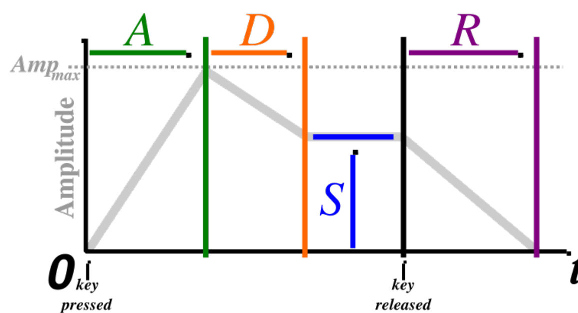


Figure 1.3[4]: Attack, decay, sustain and release time of a sound event

4 The figure is taken from http://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope

The onset time of a stroke refers to the beginning of the attack phase or to a moment during it. The audio onset detection is an active research area[5]. There is a distinction between the physical and the perceptual onset time. The physical onset refers to the starting point of the attack time phase, while the perceptual onset is an attempt to define it in line with the human perception, as the subjective time that a listener first notices a sound event. In [4] the perceptual onset time is considered to occur when the signal reaches a level of approximately 6-15 dB below its maximum value. There are some instruments, for instance the flute, whose attack time is very long and the distinction between the physical and the perceptual onset time makes sense. However, in case of percussion instruments' sounds the time between zero and maximum energy of a stroke is almost instantaneous [2]. In [5] it is stated that the difference between the physical and the perceptual onset times in percussion sounds is on the order of magnitude of a few milliseconds, while it could be as much as 50-100ms for a note bowed slowly on a violin.

1.3 Thesis' objectives and structure

This project's first objective is the literature survey on methods that could locate the onset times of strokes and classify them, that is find the specific drum/cymbal that was hit. Simultaneous strokes on two or more instruments are also taken into account. As it is presented in the next chapters, the implemented system's temporal resolution is approximately 10ms. Given that the difference between the physical and the perceptual onset times is on the order of few milliseconds, it is clear that this distinction cannot be reached by it. Beyond that, the transcription algorithm inevitably recognises the onset times of the strokes with a short latency of approximately 10-40ms (see section 4.7). For instance, if the physical onset time of a stroke was t_1, then the output onset time for the recognised stroke would be $t_1 + latency$, where $latency \in [10, 40] ms$. However, since this latency applies to all sound events it can be taken into account in order to correct the output onset time value and approximate the real one.

In the vast majority of the relevant work the transcription is limited to the basic instruments of a drum kit, namely the snare drum, the bass (or kick) drum and the hi-hat. In cases where the transcription concerns more (pitched) instruments, played along with the drums, the limitation is usually even higher; only the snare and bass drums may be transcribed reliably in practice. This is not surprising since the complexity is greatly reduced by limiting the number of the target instruments (tom-toms drums and ride/crash cymbals are usually ignored, but are always part of a typical modern drum kit). Testing with more than 3 instruments revealed that the worse results were due to the fact that when the main regions of energy of different sources overlap, as is often the case with drums such as snares and tom-toms, then it is harder for the sources to be separated correctly [2].

Nonetheless, these limited systems do deal with the most commonly occuring drums. Therefore they are a good starting point for further improvements. They could

5 See http://www.music-ir.org/mirex/wiki/MIREX_HOME

form the basis of elaborate algorithms, capable of transcribing more (percussive) instruments. The instruments of interest, also in this project, are limited to the hi-hat cymbal, snare drum and bass drum. An in-depth investigation of how such an automatic transcription algorithm could perform better than the (state-of-the-art) implementations briefly presented in the next chapter, or how it could scale up to more than three instruments, by utilising elaborate analysis and classification methods, was not part of the objectives. What is important in this work is the relatively small total number of the algorithm's computations. However, a test with tom-toms and cymbals was performed in simulation, so as to uncover few of the transcription challenges they bring.

The second objective is to implement the algorithm on an FPGA development board, so as the system to run in real-time, giving to a drummer a low-latency feedback regarding which (combination of) drums/cymbals were hit. Before the hardware implementation the development of the algorithm preceded in Matlab, where its necessary "tuning" to our needs took place. The simulation is based on test and training samples that were recorded using the same microphone and drum kits. The algorithm's core is then programmed in VHDL and tested on Terasic's DE2-70 development board.

In chapter 2 the basic approaches to unpitched percussion transcription are outlined. The simulation results and the "tuning" of the algorithm for our needs are presented in chapter 3. In chapter 4 the design and implementation of the hardware are described. The performance of the algorithm is evaluated in chapter 5.

II

Background and related work

2.1 Approaches to drums transcription

According to [2]: "approaches to percussion transcription can be roughly divided into two categories: pattern recognition applied to sound events and separation-based systems". Pattern recognition approaches try to firstly locate a stroke and then classify it, while separation-based techniques combine detection and classification into a single technique.

Depending on the application the transcription results are used for, some systems utilize preprocessing of training data in order to extract various temporal or spectral features from the specific drum kit to be used. This prior knowledge makes the transcription easier. It is more practical in interactive systems, such as electronic tutors or games, rather than in transcription of different audio recordings, where different drum kits were used.

2.1.1 Pattern recognition approaches

Pattern recognition approaches' stages are shown in figure 2.1. They begin with the segmentation of the signal into events, by either locating potential strokes and

segmenting the signal using this information, or a simpler segmentation based on a regular temporal grid. Both ways come with their own problems. Locating the potential strokes must be reliable enough so as not to misinterpret noise as a stroke. Moreover, it must be able to detect a potential stroke even when it is masked by another stroke which occured simultaneously or few milliseconds before/after. On the other hand, using a temporal grid requires the calculation of the appropriate fixed grid spacing, which has to be related to the fastest "rhythmic pulse" present in the signal. Therefore it is applicable only to cases where the tempo of a music track is known, or can be easily found.

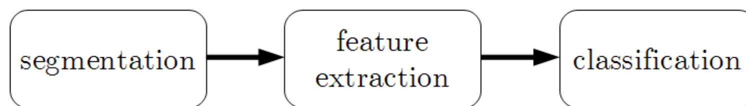


Figure 2.1: The three stages of pattern recognition approaches

After the signal's segmentation feature extraction follows, for each one of the segments. Based on the feature values the classification algorithm classify the segments into the proper category; that is recognize the source instrument if there was indeed a stroke detected. Mel-frequency cepstral coefficients (MFCCs) are widely used both in speech and music recognition. The computation of the MFCCs is based on frequency bands which are equally spaced in the mel scale, rather than being linearly spaced. The mel scale is a scale of pitches judged by listeners to be equal in distance one from another [11]. Other common features are the bandwise energy descriptors, where the frequency range is divided into few frequency bands and the energy content of each is computed and used as a feature, as well as the spectral centroid, spread, skewness and kurtosis [2]. Less often time-domain features are used, such as the temporal centroid and the zero crossing rate.

The feature set could comprise many features and is generally selected through a trial and error procedure. The properties of the input music signal help us determine which features perform better. The feature set could be also selected automatically, by testing different combinations, a procedure that requires though to find an appropriate function which evaluates their quality, which is not that simple in practice. In [2] it is remarked: "it was noticed that in most cases, using a feature set that has been chosen via some feature selection method yielded better results than using all the available features."

After the segment's features are extracted, they feed the classification algorithm. The detection of a stroke in the segment may be necessary, before classifying it into the recognised (combination of) instrument(s). Classification approaches are divided into two categories; those which try to detect the presence of any given drum/cymbal separately, so that simultaneous multiple strokes are recognized as the sum of individually recognised strokes on one instrument, and the ones which try to recognize directly each different combination of simultaneous strokes. Few examples are the decision-trees (sequences of questions whose answers determine the excluded possible results and the next questions), or methods which compare the analysed data to stored fixed data – outcome of a training

procedure (for instance k-nearest neighbors or support vector machines – SVMs). None of these methods seem to perform clearly better than the others, so some advanced techniques and higher-level processing have been incorporated to increase the performance, such as language modelling with explicit N-grams, hidden Markov models, or choosing the best feature subset dynamically [12].

2.1.2 Separation-based approaches

The implemented algorithm, which is based on the non-negative matrix factorization (NNMF), belongs to the separation-based (or source-separation) approaches. The independent subspace analysis (ISA) and the non-negative sparse coding (NNSC) are similar methods. What makes them attractive is that they analyze a single-channel signal in such a way that they may separate intrinsically the different instruments, or generally the different sound types of interest. They output distinct streams, called components, which make the transcription easier in case they are musically meaningful. In 2.1.2.1 it is discussed what "musically meaningful" could mean, in other words how components are associated with the different sources of the sounds. What follows in 2.1.2.2 is a presentation of the input signal's data representations. The approximated factorisation of the signal, which results to the source-separation itself, is presented in 2.1.2.3. In the last subsection a widely used audio onset algorithm is outlined.

2.1.2.1 Sources and components

Separation-based algorithms are meaningful in single-channel signals obtained by one microphone or by mixing down more microphones (combine them into one signal), as figure 2.2 depicts. Single-channel (mono) and two-channels (stereo) signals are usually used in practice as inputs. In case of multi-channel recordings, where the signal of each instrument has its own channel, the separation-based techniques are not preferred[6]. The use of multiple microphones itself separates the music signal in regard to the different instruments-sources.

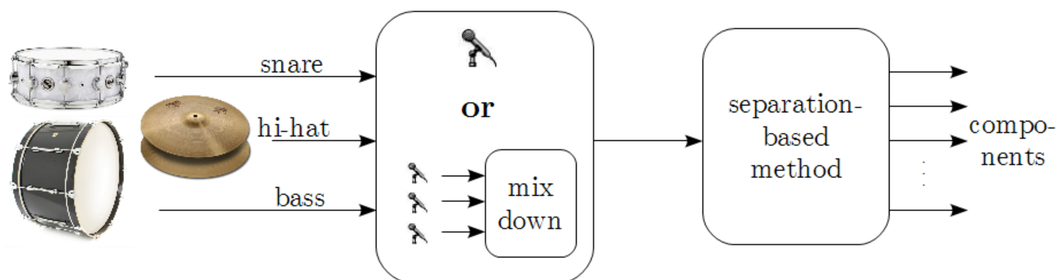


Figure 2.2: A single channel's signal is usually the input of separation-based approaches, while the output components do not necessarily correspond to one and only one of the sources, unless we force it

6 An exemption is the independent component analysis (ICA), that requires at least as many channels as there are sources [13]

Source-separation may not refer only to cases where each instrument is a single source. Depending on the application, a transcription could require, for instance, twenty violins playing simultaneously in an orchestra to be assigned to a single source. Another one may require each different violin to be considered as a single source, while for the needs of a third one the assignment of one source to each one of the equally-pitched notes could be needed, regardless of the specific violin the sound originated from.

The algorithms referred to above could be designed to output meaningful musically streams, but in order to achieve that it is necessary to utilise prior knowledge regarding the instruments' sound properties. Otherwise the separation is "blind", in the sense that it is unknown in advance how the separated components relate to the musically meaningful sources. Of course, there may be applications whose properties or careful "tuning" of their parameters, combined with the properties of the signal to be transcribed, lead to the "blind" separation being musically meaningful. But this is usually the case when the total number of components is small and the various instruments' sounds spectrums do not extensively overlap. If NNMF, ISA, or NNSC were applied without forcing any predetermined separation scheme, the resulting components would be determined by the specific algorithm's properties, the properties of the input signal and the total number of components. The latter is usually our only influence on the resulting separation.

Hypothetically the transcription of music played by three instruments is the objective; one bass drum, one guitar and one piano played along. Let the number of components, C, of each source be equal to one, resulting to total number of components equal to three. Then, after applying a separation-based method the separation result could be:

- one component which "captured" the bass drum and few of the piano's low frequency notes,
- another component which "captured" most of the guitars's notes,
- and a third component "capturing" all the rest sound events.

Depending on the value of C and the signal's and algorithm's properties any separation result is possible, even the preferred one. If the number of components of each source is increased, then the possibility of a musically meaningful "blind" separation is dramatically decreased.

Therefore further processing is necessary in order to find out which source the components refer to. An unsupervised classification algorithm could be used in order to interpret them. Alternately, prior knowledge about the sources could be utilised, so as to classify the components. However, forcing a predetermined separation scheme is easily achieved, by training the algorithm to our specific instruments' sounds, a procedure that is described in 2.2.2 for the NNMF case.

2.1.2.2 Input data representation

Short-time signal processing of possibly overlapping frames (or segments) of the signal is utilised to get the input signal's data representation. Each frame's duration is on the order of magnitude of few milliseconds, usually around 5-20ms, since percussion sounds have very short duration as it was previously mentioned in 1.2. Similarly to what was stated above regarding the pattern recognition approaches, the signal may be processed both in time and frequency domains.

Time-domain ones are more straightforward to compute and there is no loss of information, like after applying a transform in the frequency-domain. As it is stated in [13] the problem with time-domain representations is that "the phase of the signal within each frame varies depending on the frame's position". This means that all the possible different positions of a sound event inside a frame must be taken into account, which is not practical.

Regarding frequency-domain representations the discrete Fourier transform (DFT) of each frame is computed. Let the frame's length be N samples, that came of a sampling rate of f_s Hz. Then the DFT output (spectrum) consists of N complex values, each corresponding to a different frequency, linearly spaced apart by f_s/N in the range $[0, f_s]$. For 2048 samples and 44.1kHz the frequency resolution is approximately 21.5Hz. By considering the magnitude or the power spectrum, the phases of the complex-valued transform are discarded, eliminating the phase-related problems of time-domain representations. Magnitude spectrum refers to the case where the magnitude of DFT's complex-valued output is used, while power spectrum refers to the case where the squared magnitude is used. The concatenation of the frames' magnitude/power spectrums let us visualize the music signal (spectrogram); for instance, in figure 2.3 the blue colour means that the power has a small value, while the red colour means it has a large one. In cases that high frequency resolution is not needed (such as in our case), the magnitudes or the powers are usually summed up in frequency bands, whose width and partitioning depends on the application. The less bands used, the coarser the frequency resolution becomes.

The linear summation of time-domain signals does not imply the linear summation of their magnitude or power spectra, since phases of the source signals affect the result. When two signals, $y_1(t)$ and $y_2(t)$, sum in the time domain, their complex-valued DFT outputs sum linearly: $X(k) = Y_1(k) + Y_2(k)$, but this equality does not apply for the magnitude or power spectra. However, if the phases of $Y_1(k)$ and $Y_2(k)$ are uniformly distributed and independent of each other, we can write [13]:

$$E\{|X(k)|^2\} = |Y_1(k)|^2 + |Y_2(k)|^2$$

where $E\{\}$ denotes the expectation. This means that in the expectation sense we can approximate time domain summation in the power spectral domain, a result which holds for more than two sources as well. Even though magnitude spectrogram representation has been widely used and it often produces good results, it does not have similar theoretical justification [13].

2.1.2.3 The approximated factorisation

NNMF, ISA and NNSC assume that the input signal's spectrogram X results from the superposition of k spectrograms Y_j of the same size (k is the total number of components and $j=1,2,\ldots,k$). The size of X is $m \times n$, where m is the number of frequency bands and n is the total number of time frames. Each frequency band contains the sum of magnitudes/powers of all the frequencies inside the band's range. In figure 2.3 the spectrogram of a drum rhythm is illustrated. The spectrum of the *i-th* frame, X^i, is equal to Y^i, where $Y^i = Y_1^i + Y_2^i + \ldots + Y_k^i$.

Further, it is assumed that each one of the spectrograms Y_j can be uniquely represented by the outer product of a frequency basis vector b_j (of length m) and a time-varying gain g_j (of length n) [2]:

$$X = \sum_{j=1}^{k} Y_j = \sum_{j=1}^{k} b_j g_j^T$$

If each basis vector, b_j, captures one source's features, then the corresponding time-varying gain vector, g_j, can be translated as the level of the contribution of this source to each frame's content. Figure 2.4 illustrates the vectors b_j and g_j for the same rhythm's input signal, which comprise only the three instruments of interest (snare, bass and hi-hat). The frequency range is [0, 22.05kHz] and is partitioned into 25 bands (the first 25 critical bands – see 3.4.3). Note that the frequency bands, fb, are not linearly spaced (although they are depicted like they were). Instead, the first ones are narrow (fb_1=[0,100Hz), fb_2=[100Hz,200Hz), ..., fb_6=[630Hz,770Hz), ..., fb_{10}=[1.08kHz,1.27kHz)), while the last ones are very wide (fb_{24}=[12kHz,15.5kHz) and fb_{25}=[15.5kHz, 22.05kHz)). It is clearly shown in the figure that hi-hat's basis vector has large values only in the high frequency bands (fb_{21}-fb_{25}), bass' basis vector only in the low frequency bands (fb_1-fb_2), while snare's one both in low frequency (fb_3-fb_8) and high frequency bands (fb_{22}-fb_{23}).

Ideally all of g_j's local maxima would mean that a stroke on the specific source did occur at maxima's frames. But this is not the case since strokes on specific sources (or, more often, combinations of simultaneous strokes on two or more sources) may result in local maxima presence in another source's gain vector, without any stroke occuring in the latter source. Depending on the algorithm they could be misinterpreted as recognized strokes (false onset detection). In the example of figure 2.4 this is the case for false snare onsets at every stroke on the bass drum, and vice versa, for false bass onsets at every stroke on the snare drum. However, the amplitude of the false onsets' local maxima is considerably lower, allowing the use of a threshold in order to distinguish the false and the correct onsets. This is why further processing, usually inspired by Klapuri's onset detection algorithm, is usually needed, in order to detect the real strokes out of the time-varying gain vectors.

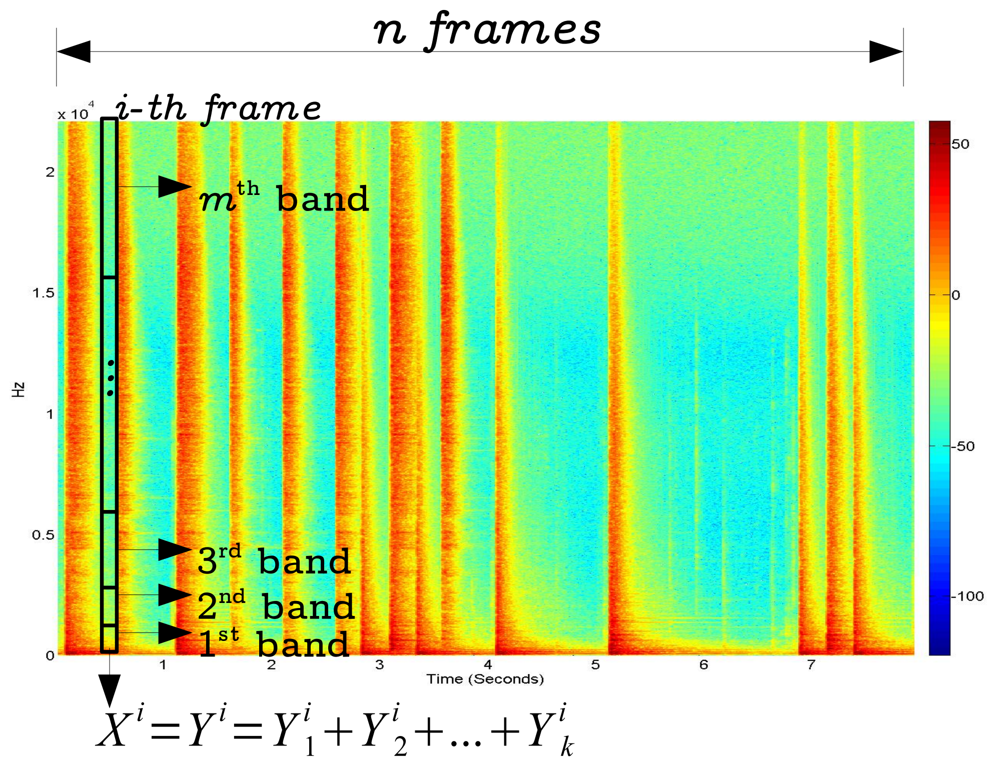


Figure 2.3: The input signal's spectrogram is assumed to be equal to the superposition of k spectrograms

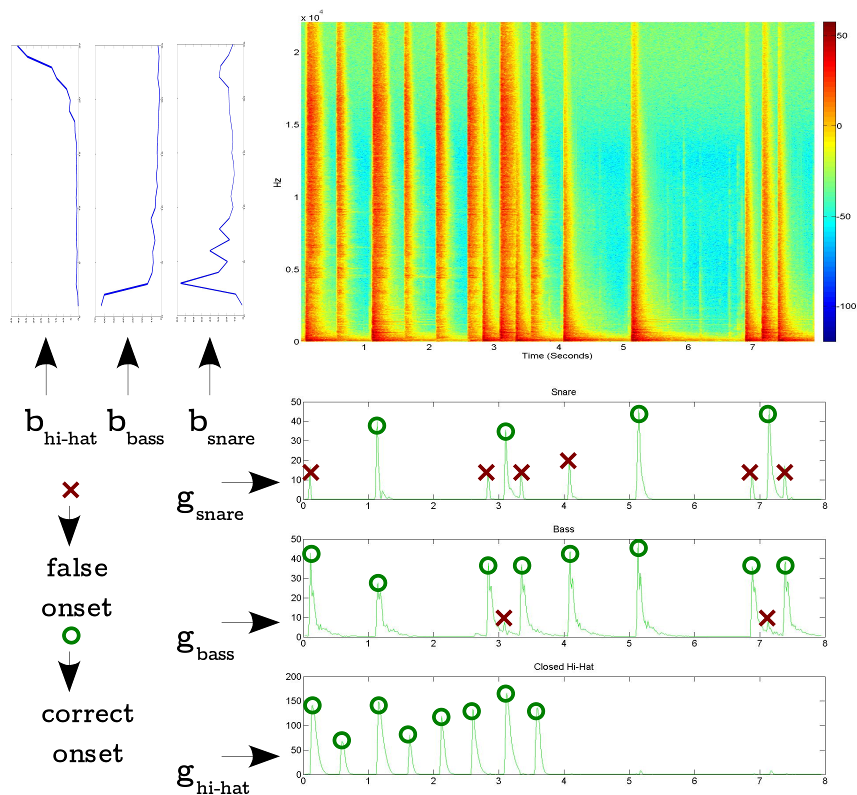


Figure 2.4: The basis vector, b, and the time-varying gain vector, g, for each component

2.1.2.4 Klapuri's audio onset detection algorithm

Many algorithms that detect the onsets of sound events are based on such an algorithm proposed by Klapuri in [3]. The overview of Klapuri's algorithm is illustrated in figure 2.5. Its input could be any kind of polyphonic music, containing many instruments, and its objective is the detection of the starting point (onset) of each single sound event. Depending on the application its output may be the input of a classifier, which recognizes to which instrument each onset corresponds to.

After the signal is normalized to 70dB level it is divided into 21 non-overlapping frequency bands. Then the algorithm detects onsets in every band, determining their time and intensity. The time detection is based on the calculation of the first order difference function of each band's logarithm. The intensity is taken from the first order difference function of the band's signal multiplied by the band's center frequency. Detected band's onsets that are closer than 50ms to another onset of larger intensity are ignored. The various bands' results are combined in the last phase of the algorithm and onsets closer than 50ms to a more intense one are again ignored. Finally, a threshold value distinguishing the true from the false onsets is determined and the algorithm yields its final results.

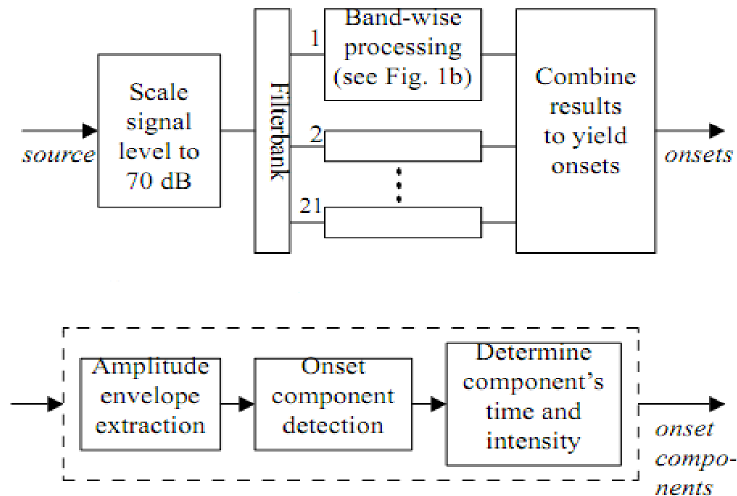


Figure 2.5 (taken from [3]): Klapuri's system overview (top) and processing at each frequency band (bottom)

2.2 Non-Negative Matrix Factorisation based approaches

When the magnitude or power spectrograms are used, by definition the basis matrices are non-negative. Some of the separation-based algorithms do not guarantee the non-negativity of both the basis and the time-varying gain matrices, producing negative values as well. Apart from the fact that there is no physical interpretation of these negative values, there is also another reason that non-negativity is a useful property. In [13] the following important conclusion is stated for NNMF, a method which guarantees the non-negativity of both matrices: "it has turned out that the non-negativity

restrictions alone are sufficient for the separation of the sources, without the explicit assumption of statistical independence".

NNMF has been used for feature extraction and identification in text and spectral data mining, as well as in data compression applications. Its principles were originally developed by Pentti Paatero in '90s (positive matrix factorization), but has been widely known after Daniel Lee and Sebastian Seung's work in '00s. In [14] two computational methods for the factorisation are proposed by the latter. The NNMF problem can be stated as follows:

[NNMF problem] Given a non-negative matrix $X \in R^{m \times n}$ and a positive integer $k < min\{m, n\}$, find non-negative matrices $B \in R^{m \times k}$ and $G \in R^{k \times n}$ so that:

$$X \approx B \cdot G$$

The product $B \cdot G$ is called a factorisation of *X*, although *X* is not necessarily equal to it. In fact, it is an approximate factorization of rank at most *k*. According to [15]: "an approximate decision on the value of *k* is critical in practice, but the choice of *k* is very often problem dependent. In most cases, however, *k* is chosen such that $k \ll min\{m, n\}$ in which case $B \cdot G$ can be thought of as a compressed form of the data in *X*".

NNMF is applied in the following manner: given a set of multivariate *m*-dimensional data vectors, the vectors are placed in the columns of a $m \times n$ matrix *X*. After the approximate factorisation of *X* into the matrices *B* and *G*, $X \approx B \cdot G$ can be rewritten column by column as $x \approx B \cdot g$, where *x* and *g* are the corresponding columns of *X* and *G*. In other words, each data vector *x* is approximated by a linear combination of the columns of *B*, weighted by the components of *g*. Therefore, *B* can be regarded as containing a basis that is optimized for the linear approximation of the data in *X*.

2.2.1 Update rules and cost functions

The matrices *B* and *G* are calculated by applying iteratively update rules to them and evaluating their resulting updated values. There are various approaches proposed in order to do so. [15] contains an extensive survey on them, presenting rules which are optimized on different aspects, such as computational efficiency, or convergence properties and speed. One of the update rules set proposed in [14] (theorem 2) is used in this project; the multiplicative update rules for the minimisation of the divergence $D(X \| BG)$. $V_{\alpha\beta}$ denotes the element $V(\alpha, \beta)$ of a matrix V, **1** denotes an all-ones matrix of size equal to the size of *X*, $.\times$ denotes the element-wise multiplication and ./ the element-wise division:

$$B \leftarrow B .\times \left[((X ./(B \cdot G)) \cdot G^T) ./(\mathbf{1} \cdot G^T) \right]$$

$$G \leftarrow G .\times \left[(B^T \cdot (X ./(B \cdot G))) ./(B^T \cdot \mathbf{1}) \right]$$

$$D(X \| B \cdot G) = \sum_{\alpha=1}^{m} \sum_{\beta=1}^{n} \left[X_{\alpha\beta} \cdot \log_{10}(X_{\alpha\beta} /(B \cdot G)_{\alpha\beta}) - X_{\alpha\beta} + (B \cdot G)_{\alpha\beta} \right]$$

The values of B and G are initialized with random positive values. After each iteration the new values of B and G are found, by multiplying the current values by some factor that depends on the quality of the approximation $X \approx B \cdot G$. The approximation's quality is quantified by a cost function, which is calculated after each iteration and determines when the convergence is supposed to be achieved. In our case, $D(X \| B \cdot G)$ is lower bound by zero (which it reaches if and only if $X = B \cdot G$) and reduces to the Kullback-Leibler divergence when $\sum_{ij} X_{ij} = \sum_{ij} (B \cdot G)_{ij} = 1$ [14]. In [14] it is also proved that: "the quality of the approximation improves monotonically with the application of these update rules. In practice, this means that repeated iteration of the update rules is guaranteed to converge to a locally optimal matrix factorisation".

2.2.2 Using prior knowledge about the sources

As it was explained in 2.1.2.1 the "blind" separation into the resulting components could be avoided by incorporating a training stage. If such an approach is implemented, the algorithm does not need to identify to which component(s) each source associates to after the separation, but this is predetermined. During the training, consecutive NNMFs are applied to monophonic input signals, which refer to the sounds of only one of our instruments/sources.

For simplicity the scenario where each source is described by one and only one component is again examined. If NNMF was applied to a monophonic input signal's spectrogram X_i, then the result would be the approximation $X_i \approx B_i \cdot G_i$, where $X_i \in R^{m \times n}$, $B_i \in R^{m \times 1}$ and $G_i \in R^{1 \times n}$ (m is the number of frequency bands and n the total number of frames of the input signal). The resulting basis matrix, therefore, is a single column one, which adapted to the frequency content of this specific source (see in figure 2.4 the differences among B_{snare}, B_{bass} and B_{hihat}, which are the result of the described methodology). If consecutive NNMFs were applied to each one of our sources' sounds, we would end up finding such a unique basis vector B_i for each one of our different sources. If, for instance, the number of sources was equal to 3 (as in our case, where we transcribe snare drum, bass drum and hi-hat), then after this first training stage a fixed basis matrix $B_{\text{fixed}} = [B_{snare} B_{bass} B_{hihat}]$ would be formed, where $B_i = [b_i^1 \; b_i^2 \; b_i^3 \ldots b_i^m]^T$.

The second stage of the algorithm makes use of this fixed basis matrix, by running only the multiplicative update rule for the time-varying gain G, ignoring the one for the basis matrix B_{fixed}. The input signal in this stage is the polyphonic signal, the subject of the transcription. Following the same example as above, it means that the input signal consists of any combination of simultaneous strokes on the three sources, as well as, of

course, strokes on only a single instrument. The time-varying gain G_i indicates the contribution of the *i-th* source, since the polyphonic signal is approximated by the weighted, by the values of *G*, sum of the basis vectors B_i of all the sources.

Figure 2.6 depicts this procedure in the general case, where the number of sources is equal to *s*, the number of frequency bands is *m*, the total number of frames/time windows processed is *n* and the number of components each source is represented by is *c*. The dimensions of B_i are $m \times c$, the ones of *X* are $m \times n$ and the ones of *G* $c \times n$. It is worth noting that a different *c* for each source could be chosen: $c_i = \{c_{snare}, c_{bass}, \ldots, c_{etc}\}$. *G*'s dimensions become $(s \cdot \sum_{i=1}^{s} c_i) \times n$ in this case.

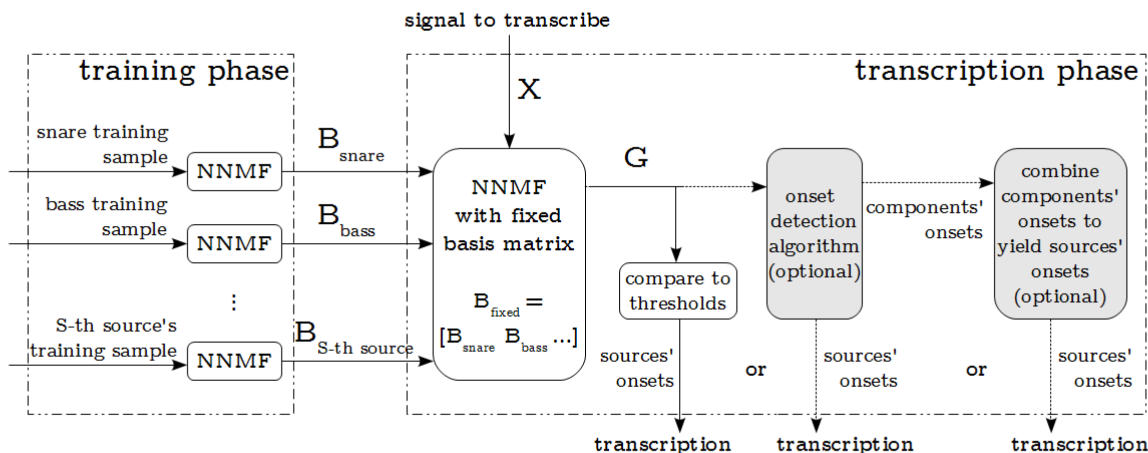


Figure 2.6: The use of NNMF with prior knowledge about the sources

After the training NNMFs which produce the *s* fixed basis matrices, another NNMF is applied, using this fixed matrix to approximate our polyphonic signal's magnitude spectrogram *X*. What follows is determined by the value of *c* and the signal's properties. If *c* was equal to one for every source, the gain matrix *G* itself could be considered as the transcription results we are looking for. In this case, the sources' onsets times are taken by the frame's position; that is for the *i-th* frame equal to $i \cdot \Delta t$, where Δt denotes the temporal resolution (10ms in our case). The only requirement for a processed frame to be considered that it contains a recognized stroke is that the corresponding source's value of *G* is greater than a threshold value (see figure 2.6). There are many ways for these thresholds to be found. The simplest one is to determine them during the training stage, by storing the maximum value, $max(G_i)$, of every row of *G*. Then, a safe threshold could be found by testing $max(G_i)/n$ for different values of $n>1$. In the next subsection, 2.2.3, an example of another method is presented, which also takes into account only the training samples but in a more supervised way. Alternatively, another algorithm, which adapts to better values by also taking into account the polyphonic signal's values of *G*, could be used.

If taking directly the sources' onset times from *G* is not possible, then what follows in order to find them is:

- the onset detection: the need for it was discussed in 2.1.2.3, and in 2.1.2.4 a common in practice algorithm for polyphonic audio signals was presented. Klapuri's algorithm is usually a common starting point for the onset detection, even in implementations where the input is not an audio signal itself, like the input G in our case.
- if $C>1$ and the sources' onset times cannot be reliably taken neither from G itself, nor from an onset detection algorithm, this means that there is contradiction in some of the components referring to the same source. Therefore, some components need to be rejected, and/or the information of multiple components need to be combined in order to find a single source's onset times (see 3.4.6 and appendix D for an example).

The 2-stages procedure described above is not the only way prior knowledge about the sources could be used in order to make the transcription easier. An even more supervised one is to manually define the basis matrix. It is applicable in cases where it is known in advance that regardless of the specific instrument's particularities, its sounds have well-known frequency content. Let's consider, for instance, a violin's music transcription case where each component of NNMF is assigned to all equally-pitched notes. Then someone could approach the problem of predetermining the separation scheme by forcing the use of fixed basis vectors, which are manually constructed based on the information about which fundamental frequencies and harmonic overtones each note is supposed to produce. The manually defined basis matrix could either remain constant, or alternatively adapt to the observed data of the polyphonic signal to be transcribed.

Whatever was presented above concerns systems which focus on only low-level recognition. However, knowledge at a higher abstraction level could be also utilized; either prior, or knowledge gained after processing the polyphonic signal. Let's consider the automatic speech recognition problem. The recognition approaches described so far resemble the case where in speech recognition each phoneme is recognized independently, regardless of its context. But this is not the case in practice, since by utilizing linguistic knowledge the performance of such systems can improve considerably [2]. By incorporating similar approaches the performance of automatic transcription systems could be also improved. Such approaches could refer to simplistic scenarios where the music player predefines some characteristics of the music he/she is going to play, like the tempo. In more advanced approaches the statistical dependencies of sound event sequences of the transcribed music could be modelled and used to correct the transcription results.

2.2.3 Relevant work on NNMF-based transcription systems

The work of Jouni Paulus and Tuomas Virtanen on an NNMF-based automatic transcription system of drums-only music ([12]) has been a guide for this project. The instruments of interest are also limited to the three basic ones: snare drum, bass drum and hi-hat cymbal. It follows the methodology presented in the previous subsection and as such it needs initial training in order to find the fixed basis matrix B_{fixed}. In their case the

training samples used were not only produced by a single instrument of each type. Instead the authors used monophonic recordings of various snare drums (and similarly bass drums and hi-hats) in order to find the basis matrix of each. Then, they averaged them creating a generic snare drum model, which was used in the same way in the next stage of their algorithm. That is it was kept fixed in order to predefine the separation scheme, and so only a multiplicative update rule for the time-varying matrix G was used for the minimisation of the divergence function.

The signal was represented by the magnitude spectrogram, while the length of each frame was 24ms with 75% overlap between consecutive frames, leading to an actual temporal resolution of 6ms. The frequency resolution was a rather coarse one, since only five bands were used (20-180Hz, 180-400Hz, 400-1000Hz, 1-10kHz, 10-20kHz).

The onset detection of each instrument is done from the corresponding row of the time-varying gain matrix G. The authors implemented an algorithm motivated by Klapuri's onset detection one. Its block diagram is illustrated in figure 2.7.

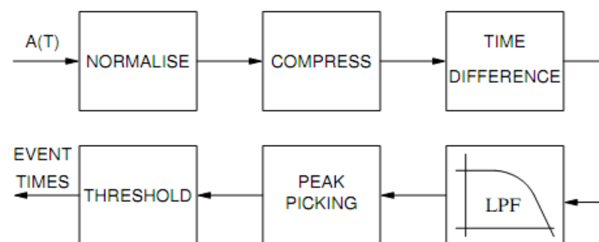


Figure 2.7 (taken from [12]): Onset detection algorithm of [12]

After the gain a_t is normalised to the range [0,1] ($0 \leq \tilde{a}_t \leq 1$), it is compressed with $\hat{a}_t = \log(1+100 \cdot \tilde{a}_t)$ and the time difference $a_t' = \hat{a}_t - \hat{a}_{t-1}$ is taken. The low-pass filter is used to reduce the low-amplitude ripple in the resulting time-difference signals and then the filtered signal is subjected to peak picking. Thresholding is used to pick only the most prominent peaks. Each instrument/source has its own threshold, which is estimated automatically during the training stage of the algorithm in the following supervised way: the training samples contain single instrument strokes whose onset times are predefined. During the training stage's NNMFs the recognized onsets are compared to the reference ones and the threshold values are chosen so that the number of undetected onsets and extraneous detections is minimised [12].

The performance of the system was evaluated with recorded material (fairly simple patterns containing only snare, bass and hi-hat) from several acoustic drum kits in different locations (medium-sized room, acoustically damped hall and anechoic chamber). The recording was made using many microphones, close microphones for snare and bass drums and overhead microphones for hi-hats. The recorded signals were mixed down to produce two different single-channel signals: one unprocessed and a "production-grade" processed one, with compression, reverb and other effects which attempted to resemble

drum tracks on commercial recordings. Two other transcription systems were evaluated with the same recordings' data, so as a comparison among them to make sense. One of them belonged to the pattern recognition approaches and is based on an SVM classifier, while the other one was also a separation-based method (PSA). The NNMF-based algorithm performed better than both of them, achieving a successful recognition of 96% and 94% of the strokes in the unprocessed and the "production-grade" processed signals, respectively. The SVM system achieved 87% and 92%, while the PSA 67% and 63%, respectively.

2.3 The Fourier transform

The ear itself is a kind of Fourier analyzer, meaning that sound is spread out along the inner ear according to the frequency. As a result, the hearing in the brain is based on a kind of "short term spectrum analysis" of the sound [8]. When someone listens to a mix of sounds which have different frequency content, the hearing process is able to separate them out. In other words, it allows us to mentally "unmix" the sounds, source-separation is an intrinsic ability of human hearing[7].

The Fourier transform can be defined for discrete or continuous, finite or infinite signals. In case of the discrete ones it is called DFT (Discrete Fourier Transform) and if it is applied to a finite sampled signal $x(n)$ of length N it is given by:

$$X(k)=\sum_{n=0}^{N-1} x(n)e^{-j\omega_k n}, \quad k\in[0,N-1]$$

X is called the spectrum of $x(n)$. DFT is a function of discrete frequency $\omega_k,\ k\in[0,N-1]$, where the frequencies $\omega_k=\frac{2\pi k}{N}$ are given by the angles of N points uniformly distributed along the unit circle in the complex plane. For the existence of the Fourier transform it is sufficient for the signal $x(n)$ to be absolutely integrable. There is never a question of existence of the Fourier transform in real-world signals, but only in idealized ones, such as sinusoids that go on forever in time [8].

If $x(n)$ is a real-valued signal, as most of the signals encountered in practice, then its spectrum is Hermitian (or "conjugate symmetric"). The real part of Hermitian spectra is even, while the imaginary part is odd [8]. In other words, the first half of the spectrum's complex-valued numbers have exactly the same magnitude with the last half of them, since their real parts are equal, while the imaginary ones are opposites.

7 Various research results of psychoacoustics (the science of the human perception of sound) have been applied successfully to audio applications

2.3.1 Window Functions

In spectrum analysis of audio signals almost always a short segment of the signal is analyzed, rather than the whole of it. This is the case mainly because the ear itself "Fourier analyzes" only a short segment of audio signals at a time (on the order of magnitude of 10-20ms) [8]. The proper way to extract a short segment from a longer signal is to multiply it by a window function. The benefit of windowing is the minimization of side lobes, which cause "spectral leakage". The side lobes are present except if the segment is periodic and an integer number of periods is the input of the Fourier transform, because the transform considers each segment as one period of a periodic signal (figure 2.8a illustrates this case).

Figure 2.8b shows an un-windowed periodic signal of a slightly lower frequency, whose segment's length is not an integer number of periods; this results to the existence of high side lobes. For the DFT it is like the two endpoints of the time-domain waveform are connected, so as 2.8a has no discontinuity at all, but 2.8b does have. These discontinuities cause the side lobes and this is what window functions attempt to correct by nullifying the first and last samples of every segment. 2.8c illustrates the spectrum of the same signal with 2.8b, but multiplied by a window function. The side lobes are clearly lower than the un-windowed case, all but the two ones at the left and at the right of the signal's frequency. These two frequencies are around 10dB higher than the un-windowed case and they are the price we need to pay for the better performance of the rest.

The selection of the proper window function requires mainly a trade-off between the level of the side lobes and the width of the main lobe and should be done in an application specific basis. The increase of the main lobe's width is what causes the adjacent bins in the left and in the right of the signal's frequency to increase, comparing to the un-windowed case, in the example of figure 2.8. The un-windowed case is identical to applying a rectangular window function, which is defined for a segment of length M as:

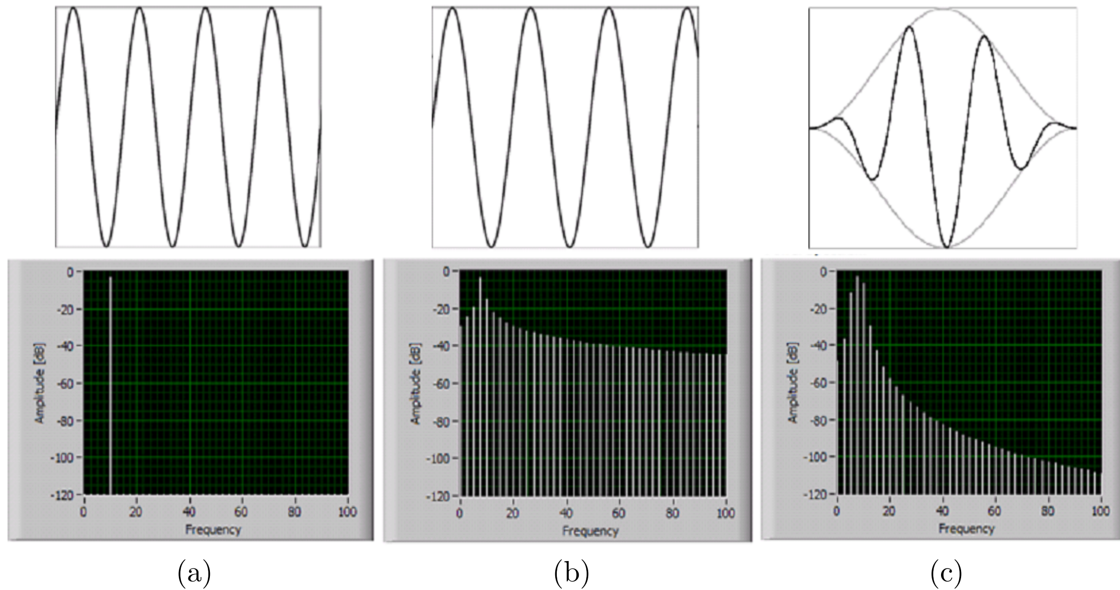
$$w_R(n)=1,\ \textit{when}\ -\frac{M-1}{2}\leq n\leq\frac{M-1}{2}\ \text{ and }\ w_R(n)=0,\ \textit{otherwise}$$

Below we present the rectangular window's properties in order to use them as a starting point on the evaluation of few representative "real" window functions' properties. The time-domain waveform and its DFT are illustrated in figure 2.9. The main-lobe width is equal to $4\pi/M$ radians per sample, so the bigger M becomes, the narrower is the main-lobe giving better frequency resolution. The first side-lobe is only 13dB lower than the main-lobe's peak and the value of M has no impact on it.

By multiplying the rectangular window by one period of a cosine the generalized Hamming window family is taken:

$$w_H(n)=w_R(n)[\alpha+2\beta\cos(\frac{2\pi n}{M})]$$

For $\alpha=0.5$ and $\beta=0.25$ the Hann window is taken, while for $\alpha=0.54$ and $\beta=0.23$ the Hamming window (illustrated in figures 2.10 and 2.11). The main-lobe's width has been doubled in both cases compared to the rectangular window case and is equal to $8\pi/M$ radians per sample, giving a coarser frequency resolution. The first side-lobe has been drastically decreased, though, being approximately 31.5dB and 41.5dB lower than the main-lobe's peak in Hann and Hamming windows, respectively. In [8] it is stated that "since the Hamming window side-lobe level is more than 40 dB down, it is often a good choice for "1% accurate systems", such as 8-bit audio signal processing systems".



(a) (b) (c)

Figure 2.8[8]: an integer number of periods results to no "spectral leakage" at all (a), while a non-integer one results to high "spectral leakage (b), which is reduced by applying a window function (c)

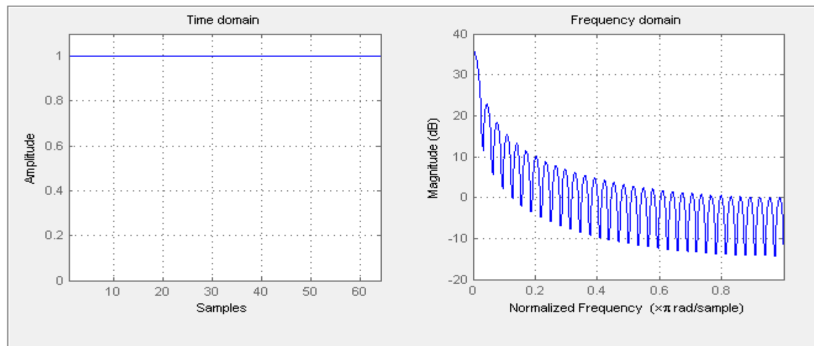


Figure 2.9: The rectangular window

8 the figure is taken from http://zone.ni.com/devzone/cda/tut/p/id/4844

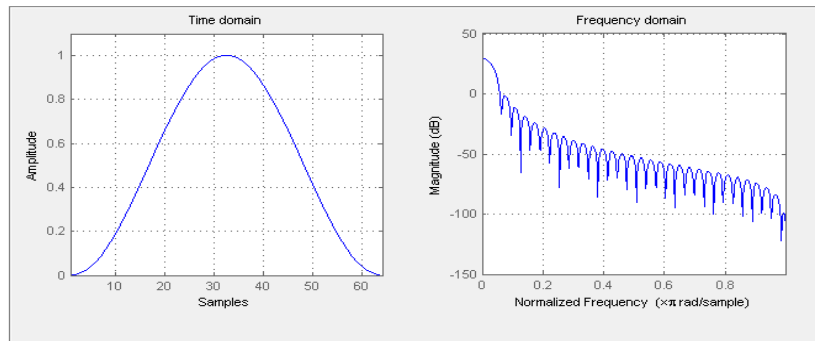


Figure 2.10: The Hann window

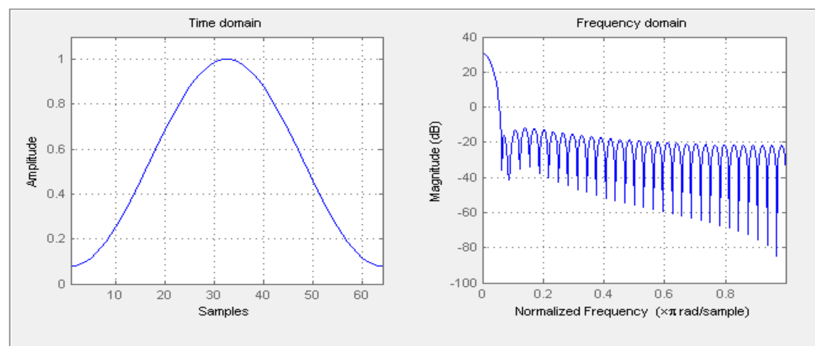


Figure 2.11: The Hamming window

Audio signals of higher quality may require higher quality window functions. The Blackman-Harris is a generalization of the Hamming family and is given by:

$$w_{BH}(n) = w_R(n) \sum_{l=1}^{L-1} \alpha_l \cos\left(l \frac{2\pi}{M} n\right)$$

For $L=4$ and $\alpha_0=0.35875$, $\alpha_1=0.48829$, $\alpha_2=0.14128$ and $\alpha_3=0.01168$ the 4-term Blackman-Harris window is taken (figure 2.12), which in expense of a quadruple main-lobe's width, compared to the rectangular window case, has a side-lobe level of approximately 92dB lower than the main-lobe's peak.

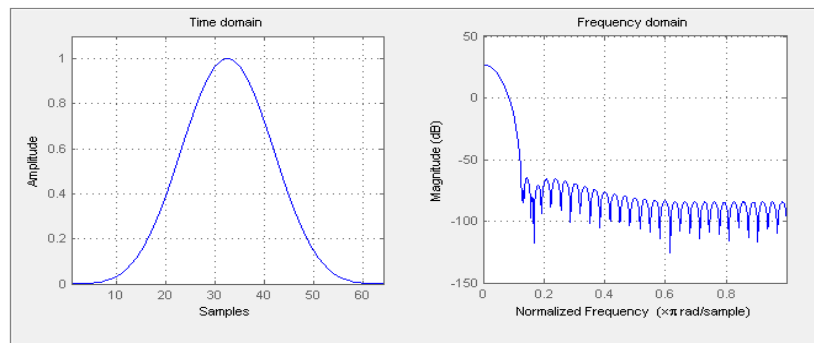


Figure 2.12: The 4-term Blackman-Harris window

There are a lot of other window functions, such as the Bartlett, the DPSS, the Kaiser, the Dolph-Chebyshev, etc. (see [8] and [9]). Depending on the signal's properties one should mainly determine the attenuation level of the side-lobes needed and the main-lobe's width that can afford to sacrifice (but as well as other less important properties of the windows, that were not referred above, such as the roll-off of the side-lobes) in order to make the appropriate selection.

2.3.2 The Short-Time Fourier Transform (STFT)

The STFT is the time-ordered sequence of spectra, taken by the DFT of short-length frames. It is used to compute the classic spectrogram, which is extensively used for speech and audio signals in general. STFT can be viewed as a function of either the frame's time or the bin's frequency. Let the frame's length be equal to *N* samples. The first frame's DFT results to the leftmost "spectrum-column" illustrated in the spectrograms. Usually the successive frames are overlapping; that is the second frame consists of the *N-R* last samples from the first one plus the *R* following ones. The *R* is called hop-size. Before the DFT is computed the frame's samples are usually multiplied by a window function; the properties of the window function determine the proper range of values of the hop-size, so that there are no "artifacts" due to the overlapping. According to [8] "the Constant Overlap-Add (COLA) constraint ensures that the successive frames will overlap in time in such a way that all data are weighted equally". Regarding the Hann and Hamming window functions any hop-size $R>N/2$ does not violate this constraint, with commonly used values being $N/4<R<N/2$. In case of Blackman-Harris window function, R should be greater than $N/3$.

Human hearing extends roughly to the range 20Hz-20kHz, although there are considerable differences between individuals. If we assumed that an audio signal has no frequencies larger than 20kHz, then according to the Nyquist-Shannon sampling theorem a sampling rate $f_s = 2\cdot 20\text{kHz} = 40\text{kHz}$ would allow the perfect reconstruction of the signal[9]. In practice sampling rates of 44.1kHz-96kHz are used in audio applications. Although any f_s greater than 40kHz is (more than) enough in order to cover the whole hearing range, often greater values are used. According to the signal's content this might lead to worse results, if proper processing of the signal was not anticipated.

The frame's length, *N*, determines the frequency resolution; that is the quantization level, or the width of each frequency bin:

$$F_{res} = \frac{f_s}{N}$$

and the temporal resolution of the STFT:

9 Actually this is true only in the idealised case, where the signal is sampled for infinite time; any time-limited sampled signal cannot be perfectly bandlimited. Therefore, in practice only a very good approximation is taken, instead of a perfect reconstruction.

$$T_{res} = \frac{N}{f_s}$$

For instance, if $f_s = 44.1kHz$ and $N=32$ then $F_{res}=1378.125Hz$ and $T_{res}=0.726ms$, while for $N=8192$, $F_{res}=5.38Hz$ and $T_{res}=185.8ms$. In figure 2.13 the spectrograms of a signal $x(t)$, composed of one out of four frequencies (10Hz, 25Hz, 50Hz and 100Hz), are illustrated for various non-overlapping frame lengths (10, 50, 150 and 400 samples/frame). It is clearly shown that by increasing N the frequency resolution gets better, while the time resolution gets worse. The definition of x(t) is:

$$x(t) = \begin{cases} \cos(2\pi \cdot 10t/s), & \text{if } 0 \leq t < 5s \\ \cos(2\pi \cdot 25t/s), & \text{if } 5 \leq t < 10s \\ \cos(2\pi \cdot 50t/s), & \text{if } 10 \leq t < 15s \\ \cos(2\pi \cdot 100t/s), & \text{if } 15 \leq t < 20s \end{cases}$$

In case of overlapping frames the hop-size R determines the actual temporal resolution of the STFT:

$$T_{actualRes} = \frac{R}{f_s}$$

For instance, if $f_s = 44.1kHz$ and R=441 then $T_{res}=10ms$, while for $R=44100$ it becomes equal to $1s$.

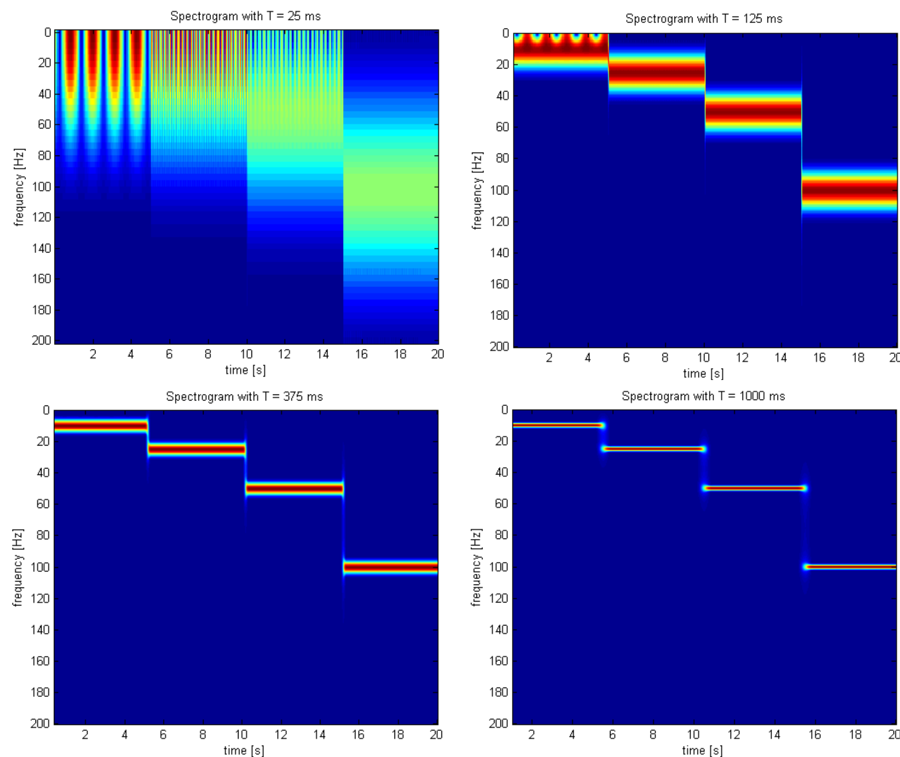


Figure 2.13[10]: Spectrograms of x(t) for N=10 (25ms) at top left, to N=400 (1000ms) at bottom right

10 The figure is taken from http://en.wikipedia.org/wiki/STFT

III

Implemented transcription algorithm and simulation

3.1 Introduction

The implemented algorithm utilises the NNMF with prior knowledge, a methodology described in 2.2.2. The reason NNMF is preferred is its lack of computational complexity, while its performance is comparable to, or even better than, more complex methods. Simulation's aim is not only to confirm that this methodology works, at least for a limited drum kit. It is also necessary in order to determine the parameters that give the best transcription results, so as to design the hardware implementation based on them, namely:

- the segmentation of the signal, that is the length of each FFT's frame which, together with the level of the successive frames' overlap and the sampling rate, gives the actual temporal resolution,
- the window function applied to the frame,
- the frequency bands' partitioning,
- the divergence threshold of the cost function, under which we consider that convergence has been achieved, and
- the number of components each source corresponds to.

3.2 Recordings

Recordings of training samples and test rthythms took place in a common, poorly soundproofed room. The drum kit was a rather old one, in bad condition, although

another, decent drum kit, was also recorded in the same room and tested without any difference at the transcription's performance.

The setup is based on a single microphone's input. Although more microphones could be used, mixed down to one signal, having only one microphone is more realistic and practical. It also suits more to the separation-based approaches, since this is usually why they are used for: "mixing up" a single-channel's signal. Moreover, using many microphones, each dedicated to only one or a limited number of instruments, makes sense in professional recordings of drums, where each channel needs separate processing. If the multiple microphones are carefully[11] setup and mixed down after the proper pre-processing, so as there is minimum interference among them, a higher quality, more "clear" input signal is taken, which makes the transcription less challenging.

3.2.1 Recording equipment

The hardware used for recording consists of the AKG's microphone Perception 170 and the Native Instuments' sound-card Guitar Rig Session I/O. Perception 170 is a small-diaphragm condenser microphone with cardioid polar pattern, suitable for acoustic instruments and percussions. Its frequency range is 20Hz-20kHz and its frequency response is illustrated in figure 3.1. At its peak at 9-13kHz the microphone barely doubles (+6dB) the magnitude of the input signal. The sensitivity of the microphone is equal to 12mV/Pa, meaning that it converts sound pressure of 1Pa to 12mV output voltage. The output is taken by a three-pins XLR connector, which beyond the ground uses both the other lines to drive the same signal, after it inverts it in one of them[12].

Guitar Rig Session I/O sound-card is external, powered by a USB port. It provides the 48V "phantom power" that the microphone needs to function, as all condenser microphones do. Its analog to digital converter can be programmed to sample with either 16 or 24-bit resolution in a sampling rate of 44.1, 48, 96 or 192kHz. Test and training samples were recorded in 44.1kHz with a 16-bit resolution.

In order to acquire the sound-card's converted signal Audacity 1.3.13 was used. Audacity is a widely used, open source tool for audio processing and recording.

---

11 In practice the proper placements, spacings and orientations of multiple microphones to achieve a high-quality recording of a drum kit is a complex task. Let a microphone A, attached to a snare drum which is 1 metre away from a high-tom drum (with another microphone B attached to it), be subject to leakage from the high-tom strokes. Since sound roughly travels 1 metre in 3ms, A will output the leakage from high-tom with 3ms of delay. As it was mentioned the strokes on percussion instruments in general have a very short attack time (on the order of a few milliseconds). The delay introduced by A would result to the "blur" of the high-tom's stroke, if it was not taken into account and properly corrected before the mixing down of the signal.

12 This way if the microphone's output is amplified by a differential amplifier, the noise voltage that was added to both signals (in the same level since the impedances at the source and at the load are identical) will be cancelled out, making the use of long cables possible in environments with high electromagnetic interference.

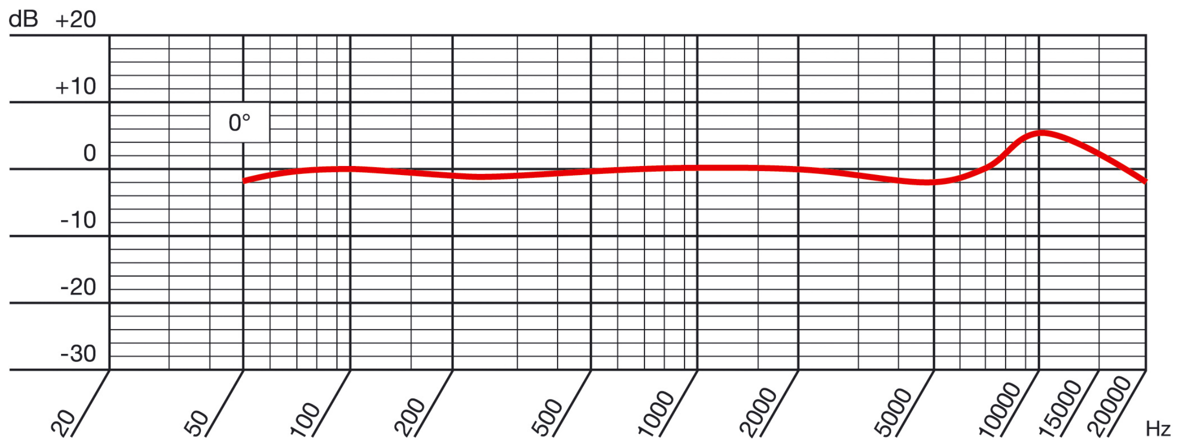


Figure 3.1[13]: The frequency curve of AKG Perception 170

3.2.2 Tempo, note's value and time signature in the common musical notation scheme

Tempo defines the speed successive notes must be performed with. It is defined as beats per minute (bpm) and if we assign to the beat one specific note value (whole note, half note, quarter note, etc) it uniquely determines the performance's speed in a common music notation scheme, like the one in figure 3.2. On the contrary, it means nothing by itself, without determining "which is the beat" among the various note values. The convention found in the vast majority of cases in practice, and also followed in this project, is quarter notes to be considered as the beats.

The *note value* denotes its duration relatively to the other notes. For example, a whole note must be played with the double duration comparing to a half note, the same with a half note comparing to a quarter note, etc. One bar (or measure) contains notes whose total duration is equal to the time signature. The successive bars are separated by vertical lines.

The *time signature* is a fraction written once in the beginning of a tablature. If it is equal to 4/4, which is the most common time signature in western music, one bar must contain notes whose total duration is equal to the duration of four quarter notes (any combination of notes whose values sum up to this duration, like just one whole note, or one half note plus one quarter note plus two eighth notes, and so on). Similarly, a tablature with time signature equal to 7/8 must contain bars whose note values' sum is equal to seven eighth note values' sum.

Figure 3.2 illustrates the main groove recorded, used for testing the algorithm in simulation. The quarter note is considered to be the beat and the time signature is 4/4. If the tempo is 60bpm then four beats, that each of bars contains, have total duration of 4

13 The figure is taken from http://www.akg.com/site/products/powerslave,id,1056,pid,1056,nodeid,2,_language,EN,view,specs.html

seconds. This means that the successive eighth notes' onset times distance is equal to 500ms and the one of sixteenth notes is 250ms. The maximum tempo that we performed and recorded is 150bpm, meaning that these distances become 200ms and 100ms, respectively. Automatic transcription systems may not allow a stroke to be recognised if it occurs before a minimum time interval passes from the last recognised stroke. In [3] the authors use such an interval of 50ms[14]. That makes the speed of our rhythms pretty challenging. Such an interval of 50ms was also used in our case, but applied only to each instrument itself, meaning that a stroke on the i-th instrument would not be recognised, if no more than 50ms passed after the recognition of another stroke on the same instrument i.

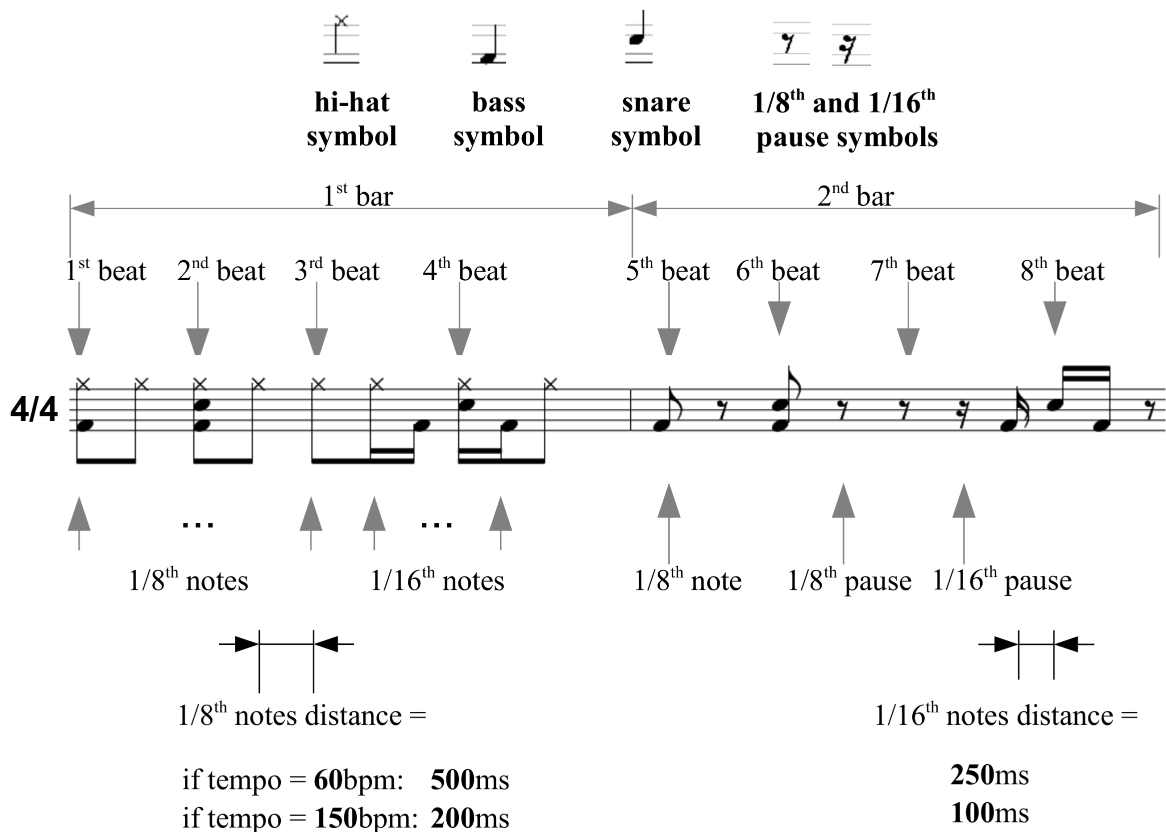


Figure 3.2: tempo, note values, time signature, bars and time difference between successive notes.

Therefore, it has been clear why the information about the onset times of the strokes, together with the information about which instruments were hit, may not be enough to uniquely determine how the notes should be written; because this also depends on the music notation scheme to be used. For example, if we were to fill just a simple time grid with the recognized strokes, then we would not need any more information. But, in order to write the notes on a common tablature, like the one in figure 3.2, it is

14 To get an idea of how restrictive this is, it is worth noting that only a few drummers in the world can play sixteenth notes on double-bass (that is they have two bass drums, one at each foot) in a greater tempo than 250bpm. This means that a right foot's bass stroke is followed by a left foot's bass stroke (and so on) with only 60ms separating the successive strokes. This speed is "insane" (more than 16 hits in one second), but still lower than what a system with 50ms limitation can handle.

necessary to know the time signature, the tempo and the note value that refers to tempo's "beat". However, these parameters' values are invariate in the vast majority of music tracks, or change only few times during them. More precisely, the "beat" is almost always assigned to the quarter note value and the time signature rarely change during the same track. The tempo, though, could change, but is usually kept invariate for many bars. In case a drummer defined the tempo, the "beat" and the time signature in advance, it would be possible for an automatic transcription system to output what he played in the classic music notation scheme, something that is beyond the scope of this project.

3.2.3 Test and training samples

Two of the test rhythms that were recorded and tested in simulation are illustrated in figure 3.3. The top one consists of only the three instruments, while the bottom one, in addition to them, contains two tom-toms (high-tom and low-tom) as well as two cymbals (ride and crash). They were recorded in four different tempos (60, 90, 120 and 150 bpm), so as to test the algorithm's performance from a relatively slow speed up to a challenging one. Figure 3.3 depicts that all possible combinations of simultaneous strokes are present in the simple rhythm, namely snare plus bass, bass plus hi-hat, hi-hat plus snare and snare plus bass plus hi-hat, together with the strokes on just a single instrument. Their sum is 7 different sound events that the algorithm should be able to distinguish.

Figure 3.3: Three-instruments rhythm (top) and seven-instruments one (bottom)

It must be clarified that "hi-hat stroke" means in our case the "closed hi-hat" type of stroke. The hi-hat is a unique type of cymbal since it consists of two metal plates, which are mounted on a stand, one on top of the other. The relative position of the two cymbals is determined by a foot pedal. When the foot pedal is pressed, then the two cymbals are held together, there is no space between them, and that is the "closed hi-hat" position. The less the foot pedal is pressed, the greater the plates' distance becomes, reaching its maximum value when the pedal is free ("open hi-hat"). The closed hi-hat stroke is one of the most important, since it produces a short length sound, unlike the other cymbals, which is usually used by the drummer to help him "count" the beats and properly adjust the timings of the strokes.

The 7-instruments rhythm does not contain all possible combinations of simultaneous strokes, since this number is large, it would make the recording complex and transcribing more than three instruments is out of this project's scope. In order to figure out how many different combinations could exist among these seven sources we need to take into account what is realistic in practice. For instance, the simultaneous strokes on three cymbals is impossible since all cymbals are hit by the hands' sticks. Actually it is only the bass drum's stroke that is driven by the drummer's foot. This limits the maximum number of simultaneous strokes to three (both hands hit a drum or cymbal and the foot also hits the bass drum – the other foot always keeps the hi-hat closed). The total number of combinations becomes equal to the sum of:

- 7 single strokes
- $\binom{7}{2}=\frac{7!}{2!(7-2)!}=21$ combinations of simultaneous strokes on two sources
- $\binom{6}{2}=\frac{6!}{2!(6-2)!}=15$ combinations of simultaneous strokes on three sources, with the bass drum always included

Beyond the rhythms that were recorded to test the transcription performance, short training samples were also recorded. They consist of successive strokes on only one instrument, with total length of 1.5s. One to eight strokes in each sample were tested, without any difference at all to the results.

3.3 Algorithm's pseudocode

The system that is illustrated in figure 3.4 was implemented and tested in Matlab. Its pseudocode follows below. The number of sources in the general case is equal to S, the number of frequency bands is M, the total number of frames/time windows is N and the number of components each source is represented with is C. The element-wise multiplication and division of two matrices is denoted by " $.\times$ " and " ./ ", respectively, and **1** denotes an all-ones matrix.

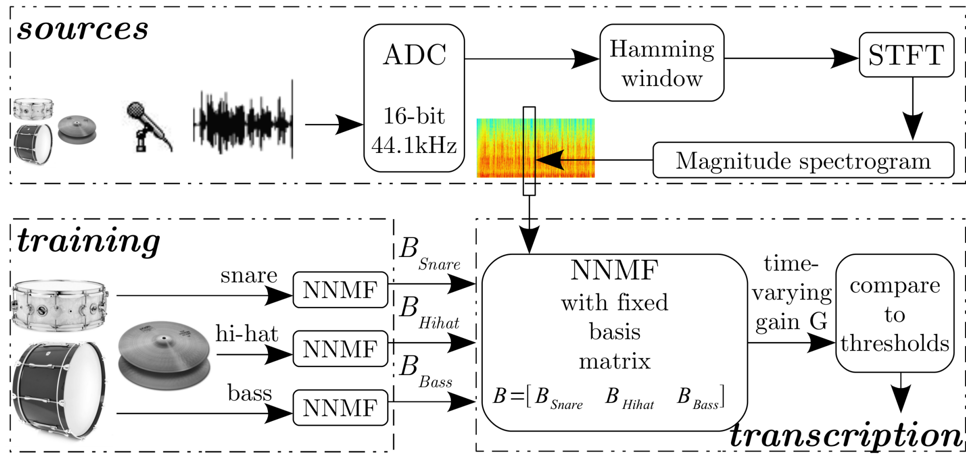


Figure 3.4: The implemented algorithm

x_{snare} ← import snare training sample
x_{bass} ← import bass training sample
x_{hihat} ← import closed hihat training sample
x_{etc} ← ...(remaining training samples)
x_{test} ← import polyphonic test signal

for every instrument $i \in \{snare, bass, hihat, \ldots\}$
    Y_i ← get STFT of windowFunction(x_i)
    X_i ← get band-wise sums of Y_i's magnitude spectrogram
    initialize matrices B_i ($M \times C$) and G_i ($C \times N_i$) to ones

    while {cost function > convergence threshold}

$$B_i^{new} \leftarrow B_i .\times \left[((X_i ./ (B_i \cdot G_i)) \cdot G_i^T) ./ (\mathbf{1} \cdot G_i^T) \right]$$

$$G_i^{new} \leftarrow G_i .\times \left[(B_i^T \cdot (X_i ./ (B_i \cdot G_i))) ./ (B_i^T \cdot \mathbf{1}) \right]$$

$$costFunction \leftarrow \sum_{m=1}^{M} \sum_{n=1}^{N_i} \left[X_i^{m,n} \cdot \log_{10}\left(\frac{X_i^{m,n}}{B_i \cdot G_i^{m,n}}\right) - X_i^{m,n} + B_i \cdot G_i^{m,n} \right]$$

    end while

    save B_i
end for

B_{fixed} <- concatenate B_i s: [B_{snare} B_{bass} B_{hihat} ...]

for n = 1 to N_{test}
    initialize matrix G^n ($S \cdot C \times 1$) to ones
    Y_{test}^n ← get STFT of the windowed n-th frame
    X_{test}^n ← get band-wise sums of Y_{test}'s magnitude spectrogram

    while {cost function > convergence threshold}

$$G^{n,new} \leftarrow G^n .\times \left[(B_{\text{fixed}}^T \cdot (X_{test}^n ./ (B_{\text{fixed}} \cdot G^n))) ./ (B_{\text{fixed}}^T \cdot \mathbf{1}) \right]$$

$$costFunction \leftarrow \sum_{m=1}^{M} \left[X_{test}^{m,n} \cdot \log_{10}\left(\frac{X_{test}^{m,n}}{(B_{\text{fixed}} \cdot G_{test})^{m,n}}\right) - X_{test}^{m,n} + (B_{\text{fixed}} \cdot G_{test})^{m,n} \right]$$

    end while

end for

3.4 Simulation results

3.4.1 Determining frame's overlapping level and length

In order to find the optimal frame's length, N, the optimal value of the actual temporal resolution, $T_{actualRes}$, must be taken into account. The actual temporal resolution depends only on the value of the hop-size, R, if the sampling rate, f_s, is constant. Since the inequality $N>R$ must hold (for $N=R$ there is no overlapping), the frame's length should be:

$$N > R = T_{actualRes} \cdot f_s$$

For f_s=44.1kHz, 441 new samples come every 10ms. A sound of a drum, or the initial phase of it in case of a cymbal, could last even less than 100ms. Therefore an actual temporal resolution on the order of 5-50ms is required, corresponding to 220-2205 samples. As it was previously mentioned in 2.3.2, the choice of the window function affects the range of R, so that the successive frames will overlap in time in such a way that sampled data are weighted equally. In case of Hann and Hamming windows a safe choice for R is given by R>N/2, while for Blackman-Harris windows by R>N/3. Therefore, if it is assumed that 220<R<2205, the possible values for the frame's size are, in case of Hamming window: 440<N<4410 while R/N>50% holds. N is usually equal to a power of two and if it is not, the edges of each frame are zero-padded in order to become so.

In figure 3.5 the transcription results for N={512, 1024, 2048, 4096} are illustrated. Table 3.1 shows the actual temporal resolution and overlapping level of each value. The frequency bands are the 25 critical ones, the divergence threshold is 10^{-4}, the number of components of each source is 1 and the input file is the rhythm of 150bpm. As N gets larger the time resolution worsens, as it is more clearly shown at the zoomed part of the hi-hat's transcription. However, a large N results to smoother transcription, with less local maxima that could be misinterpreted as onsets. It is worth noting, though, that the results are pretty close to each other and any value of N could be used.

The horizontal dashed green line defines the correct onset threshold for each source; if a value of the row of G that corresponds to this source is greater than the threshold, an onset is recognized. Each source has its own threshold value. It is not analytically computed by one of the methodologies described in 2.2.2, but rather was drawn on top of the Matlab's figures just to give an indication regarding the distances among the correct and the possible false onsets. All four values of N result to the same four false onsets, although for a larger N the magnitude is considerably smaller, at least in the case that is depicted in the zoomed hi-hat's segment. That could be explained by the higher frequency resolution that prevents a (combination of) stroke(s) to create a false onset on a source that was not hit.

N	R	Overlapping level =(N-R)/N	$T_{actualRes}$ =R/f_s
512 samples	265 samples	≈ 52%	≈ 6ms
1024 samples	441 samples	≈ 57%	10ms
2048 samples	441 samples	≈ 78%	10ms
4096 samples	441 samples	≈ 89%	10ms

Table 3.1: The actual temporal resolution for various hop-sizes and constant frame length of 4096 samples

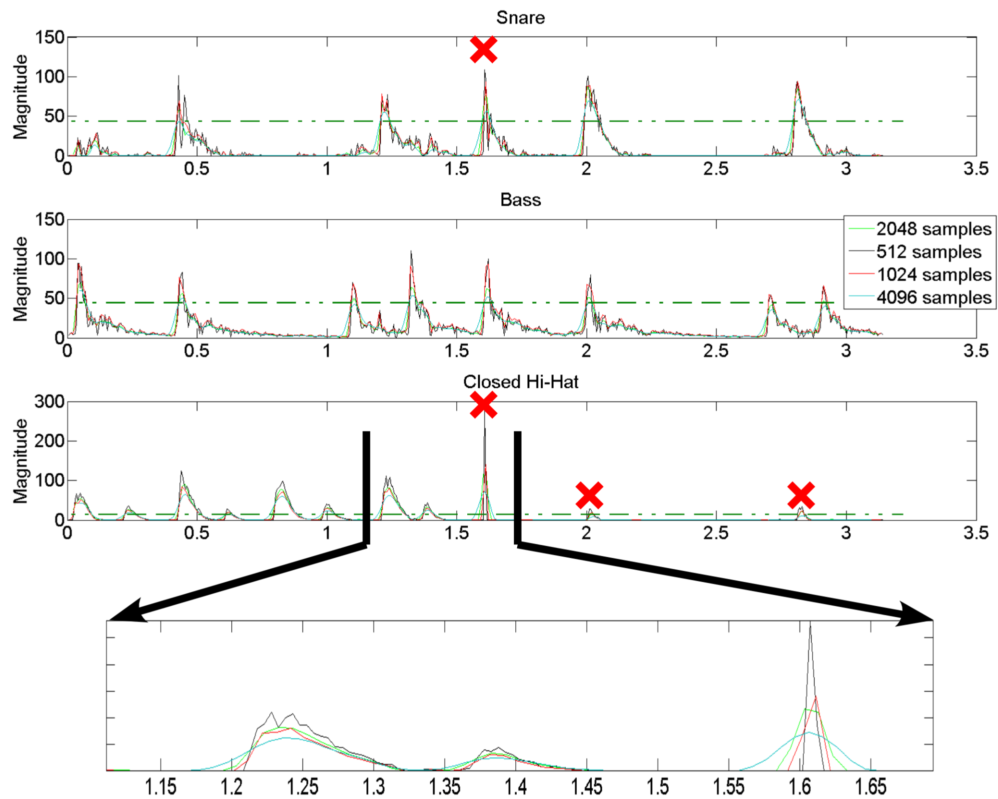


Figure 3.5: Transcription of the 150bpm rhythm for various frame's lengths

The transcription results for the 60bpm rhythm[15] and N={512, 1024, 2048, 4096} are illustrated in figure 3.6. The rest parameters are the same as above. In this case the number of false onsets is only one. It is worth noting that the 8 hi-hat strokes have, more or less, the same magnitude, while this was not the case for the 150bpm rhythm of figure 3.5. It happens simply because the recorded strokes themselves have equal intensity in the 60bpm rhythm, while every second stroke of the 150bpm is of much lower intensity. That is the usual way of playing hi-hat in high tempos and was recorded like that in order to check if different stroke dynamics are tolerated by the algorithm. At least in the closed hi-hat case, dynamics of low intensity result to low values in G; so low that if a threshold covering them had to be found, inevitably the two last false onsets of hi-hat would have been exposed.

15 The 90bpm and 120 bpm rhythms have exactly the same behavior with the 60bpm and the 150bpm, respectively, and that's why they are not presented. The rest of the tests concern only the 150bpm rhythm.

Beyond the strokes of lower intensity, more intense strokes may also cause false onsets recognition. This can be explained by their different frequency content, caused by two factors. Firstly, the physical properties of the instruments, which result to different frequency content in case of different stroke's dynamics. Secondly, the fact that all the instruments are mounted on the same rack, and hence are being vibrated even if another instrument was hit, especially for intense strokes. Appendix A contains the transcription of successive strokes of increasing intensity on each single instrument. The intensity covers a wide range of dynamics, from barely listenable strokes to unrealistically intense ones. Hi-hat does not produce much "noise" on snare and bass, but snare and bass strokes produce considerable noise on hi-hat and snare, respectively, whose magnitude is increasing for intense strokes.

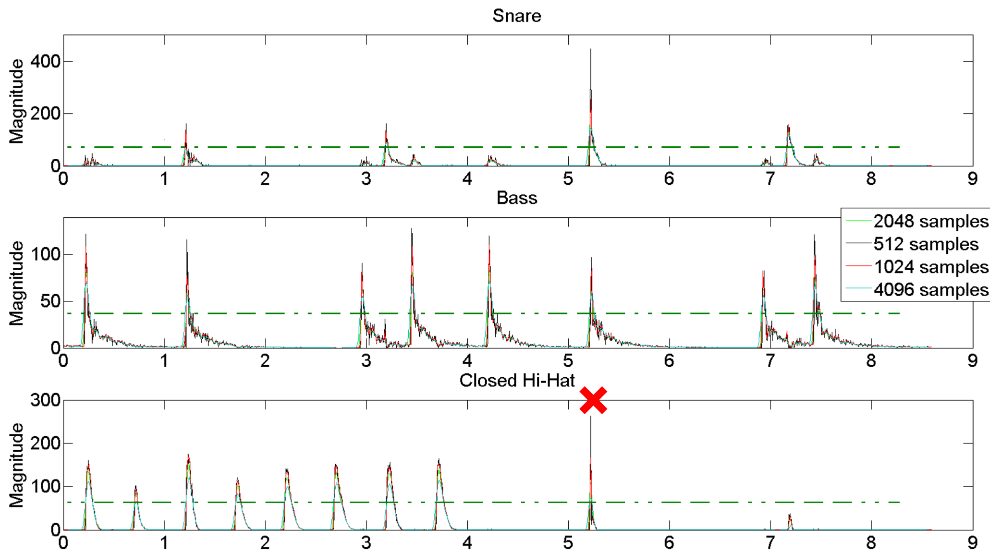


Figure 3.6: Transcription of the 60bpm rhythm for various frame's lengths

Figure 3.7 shows the impact of the actual temporal resolution to the results; for N=4096 samples, the values of R={221, 441, 661, 882, 1764, 2646} are tested. The rest of parameters are the same as above. For R=2646 the inequality N<2R is violated, but the transcription is relatively close to the lower values' ones. For 5-20ms the results are almost identical. The chosen value of $T_{actualRes}$ is 10ms. The six values of R correspond to the actual temporal resolutions and overlapping levels of table 3.2.

R	Overlapping level =(N-R)/N	$T_{actualRes}$ =R/f_s
221 samples	≈ 95%	≈ 5ms
441 samples	≈ 89%	10ms
661 samples	≈ 84%	≈ 15ms
882 samples	≈ 79%	20ms
1764 samples	≈ 57%	40ms
2646 samples	≈ 35%	60ms

Table 3.2: The actual temporal resolution for various hop-sizes and constant frame length of 4096 samples

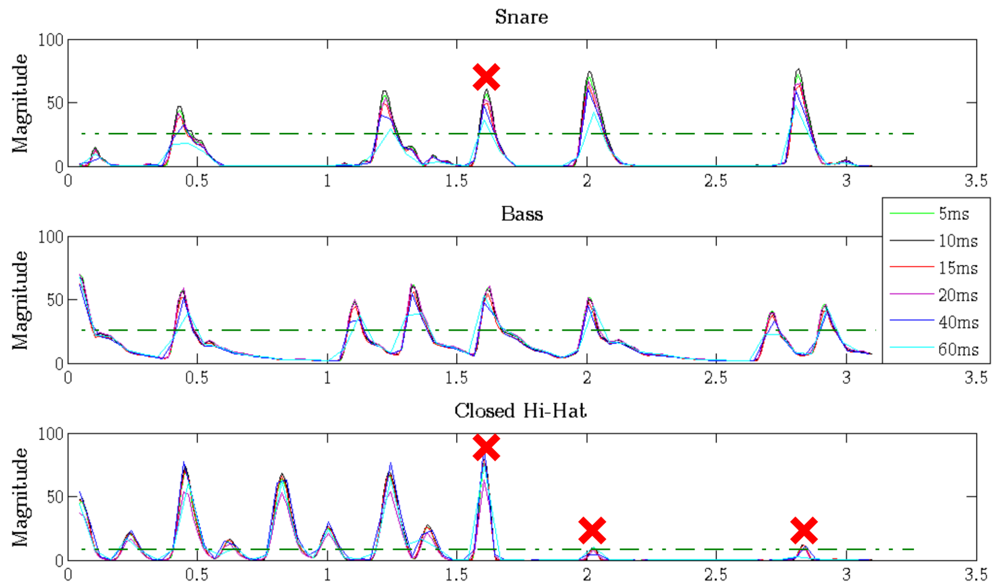


Figure 3.7: Transcription of the 150bpm rhythm for various actual temporal resolutions

3.4.2 Determining the window function

In order to examine how applying different window functions affects the results, four window functions were tested: the rectangular window, the Hann, the Hamming and the 4-term Blackman-Harris one. The frame's length is equal to 2048 samples, the hop-size is 441 samples, the frequency bands are the first 25 critical ones, the divergence threshold is 10^{-4}, the number of components of each source is 1 and the input file is the rhythm of 150bpm.

The transcription results are illustrated in figure 3.8. No window function seems to perform better than the others. Even the un-windowed case produces as good results as the windowed ones. That can be explained from the properties of the drums' sounds; since they contain frequencies in very wide ranges, the spectral leakage has no impact on the results. Although the Blackman-Harris window results to slightly larger values for G, it does so for both the correct and the false onsets, therefore that cannot be considered as an advantage. However, the Hamming window was chosen to be implemented, since it is the most common choice in practice.

3.4.3 Determining the frequency bands

In this subsection different partitioning schemes of the frequency range are tested. So far, only the 25 critical bands were used. They are based on the Bark frequency scale, which "ranges from 1 to 24 Barks, corresponding to the first 24 critical bands of hearing. The published Bark band edges are given in Hertz as [0, 100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500,

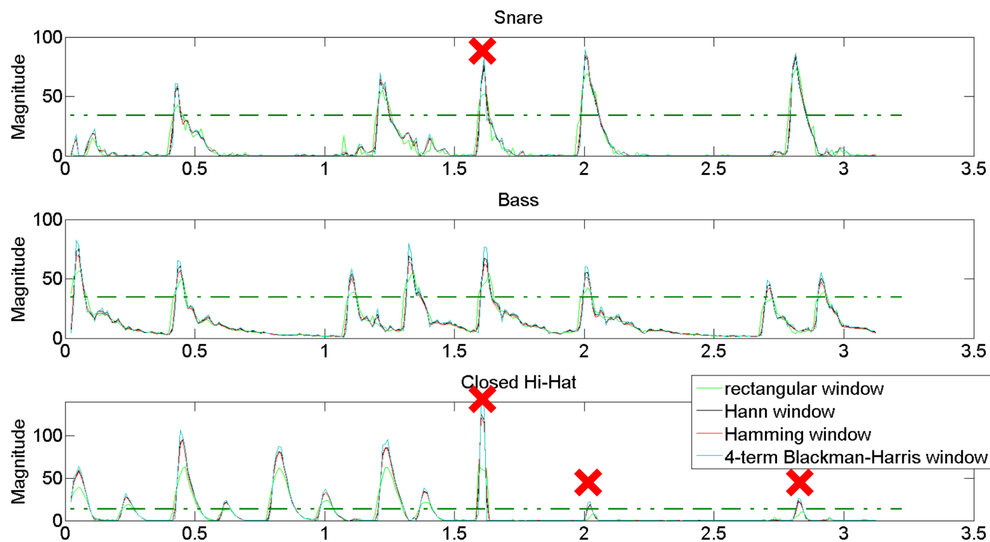


Figure 3.8: Transcription of the 150bpm rhythm for various window functions

12000, 15500]" [16]. A 25th band was appended, ranging from the 25th Bark edge, 15500Hz, to the half of our sampling rate, 22050Hz.

Beyond the 25 critical bands, the other band partitioning schemes tested were the 5-bands scheme of [12], defined by the following band edges [0, 180, 400, 1000, 10000, 22050], the 9-bands scheme defined by [0, 100, 200, 300, 400, 500, 1000, 5000, 10000, 22050] and two schemes of 128 and 512 linearly spaced bands, corresponding to equal bands of approximately 172Hz and 43Hz, respectively. The frame's length is equal to 2048 samples, the hop-size is 441 samples, the Hamming window is used, the divergence threshold is 10^{-4}, the number of components of each source is 1 and the input file is the rhythm of 150bpm.

Figure 3.9 shows that the frequency resolution is not an issue for the given signal's properties. The clear differences in the frequency content among the strokes on snare, bass and hi-hat make it possible for a 5-bands scheme, that contains band widths on the order of 180-12050Hz, to perform as good as a linearly spaced one, with band width equal to 43Hz.

This is also clearly depicted at the values of the fixed matrices B that came of the training NNMFs in the 5-bands case, shown in table 3.3. The bass has its energy mainly in the first band, snare in the fourth and hi-hat in the fifth one. We might be able to reduce the number of bands to three without considerably affecting the results, although that would introduce very poor resolution in case more instruments need to be transcribed. On the contrary, the 25 critical bands were used, allowing more instruments, which demand finer resolution, to be introduced.

band (Hz)	snare	bass	hi-hat
0-180	5.27	125.84	1.39
180-400	36.54	57.25	2.45
400-1000	41.38	60.67	7.03
1000-10000	95.80	55.54	61.90
10000-22050	33.72	10.09	94.66

Table 3.3: The values of the fixed matrix B for the 5-bands scheme

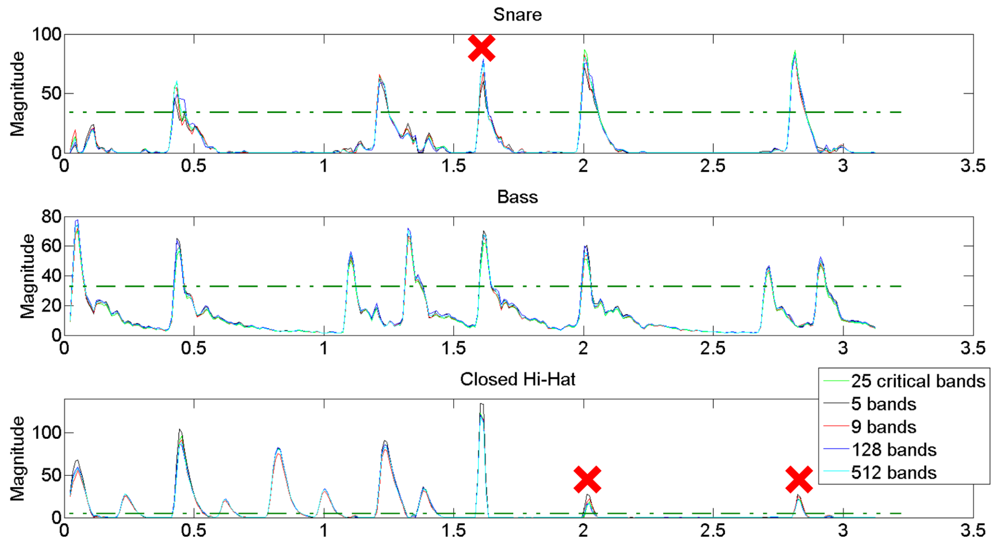


Figure 3.9: Transcription of the 150bpm rhythm for various frequency bands schemes

3.4.4 Determining the convergence threshold

NNMF runs iteratively until the product of the two approximated matrices, B and G, is close enough to X, the magnitude spectrogram input matrix. The quality of the approximation is quantified by the cost function in the end of each iteration. If the difference of the updated cost function's value from its previous value is less than the divergence threshold, then the algorithm is considered to have been converged. In figure 3.10 the transcription results for six values of divergence thresholds are illustrated. Frame's length is 2048 samples, hop-size is 441 samples, Hamming window is used, frequency partitioning of 25 critical bands, each source is represented by a single component and the input file is the rhythm of 150bpm. The results are very close to each other, except in the case of threshold equal to 100 (see the zoomed spots, all possibly false onsets).

Table 3.4 depicts the average and maximum numbers of iterations for the same values, showing that in the case of threshold being equal to 100, only 2 iterations of the update rules took place in average. The table's values concern the NNMF with the fixed

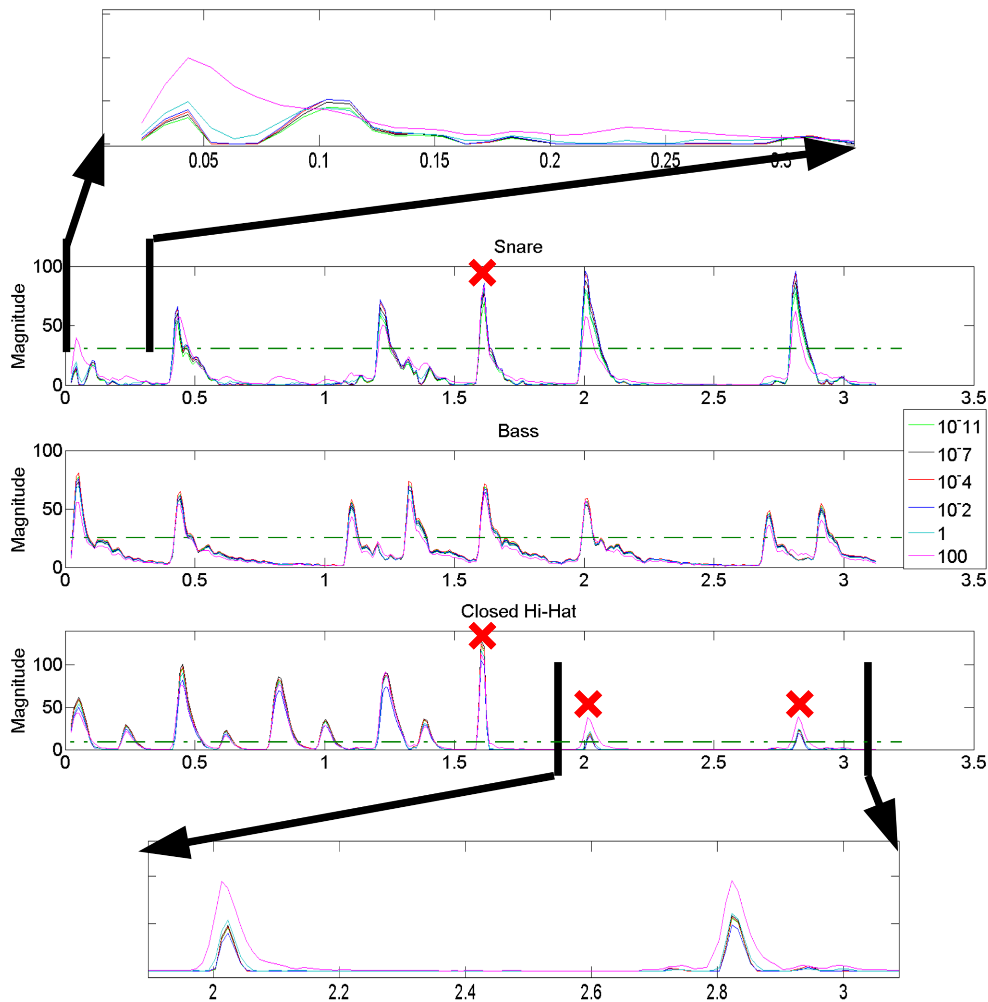


Figure 3.10: Transcription of the 150bpm rhythm for various divergence thresholds

basis matrix, since this is the only NNMF that needs to run in real-time in the next chapter's hardware implementation. The training NNMFs have no real-time requirements and as such could use a lower threshold value. That is not necessary, though, since alike the fixed basis NNMF they do not need many iterations in order to find good values for the fixed matrices. Therefore, the same divergence threshold was used for both the training NNMFs and the fixed basis one. Table B.1 in appendix B shows the snare's basis matrices for three thresholds. The values are very close even in the extreme cases.

3.4.5 Determing the number of components per source

Each source could use more than one component, as it was discussed in 2.1.2.1. Figure 3.11 depicts the case where each source is represented by two components. Frame's length is 2048 samples, the hop-size is 441 samples, Hamming window is used, the frequency bands are the 25 critical ones, the divergence threshold is 10^{-4} and the input file is the rhythm of 150bpm. The two components are colored green and black, while a safe threshold is shown in red color. The introduction of the second component has improved

divergence threshold	average number of iterations	maximum number of iterations
10^{-11}	654	20940
10^{-7}	304	3051
10^{-4}	110	604
10^{-2}	39	159
1	10	30
100	2	5

Table 3.4: The average and maximum numbers of iterations

considerably the transcription results. If only one component is taken into account (the highlighted ones – black colored for the snare and green colored for the bass and hi-hat), then a threshold can be found so as only one false onset is recognized, rather than four.

Introducing more components does not succeed in eliminating that false onset. As it was mentioned before, the usage of multiple components per source requires an algorithm that finds which is the "correct" one. In case the number of components is large (more than 4 in our case), inevitably the information of more than one components must be combined in order to find all the correct onsets, adding more complexity to this algorithm. For simplicity, our implementation concerns the one component per source case.

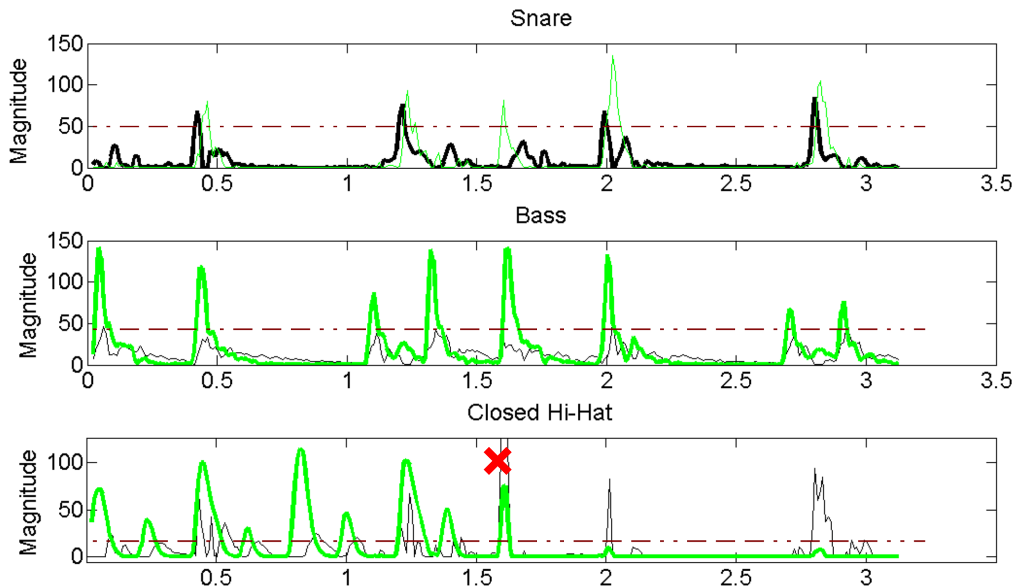


Figure 3.11: Transcription of the 150bpm rhythm for two components per source

3.4.6 Testing seven-instruments rhythms

When more instruments are introduced the transcription's performance worsens. The second rhythm's transcription is illustrated in figure 3.12, showing the results in case two tom-toms, one ride cymbal and a crash cymbal take part in the rhythm. Frame's length is 2048 samples, the hop-size is 441 samples, Hamming window is used, the frequency bands are the 25 critical ones, the divergence threshold is 10^{-4} and the input file is the rhythm of 60bpm. In appendix C the spectrograms of the seven instruments' training samples are illustrated. Only 12 out of the 46 realistic combinations of strokes are present in the rhythm, as it was mentioned in 3.2.3. Beyond the snare and bass drums which perform with 100% success rate, the rest instruments' rate becomes 50%.

Testing for more than one components per source reveals the usefulness of multiple components. In appendix D the transcription of the same rhythm is illustrated, for seven components per source. It turns out that by combining 2-4 out of the 7 components and ignoring the rest, a 100% success rate can be achieved for every instrument, but the hi-hat. The combination in most of the cases regards just the addition of the 2-4 components' values. In the case of crash, though, another type of combination may be more uselful; that is recognizing the onset based on the value of one component only if it is followed by a specific behavior of a second component (see figure D.7).

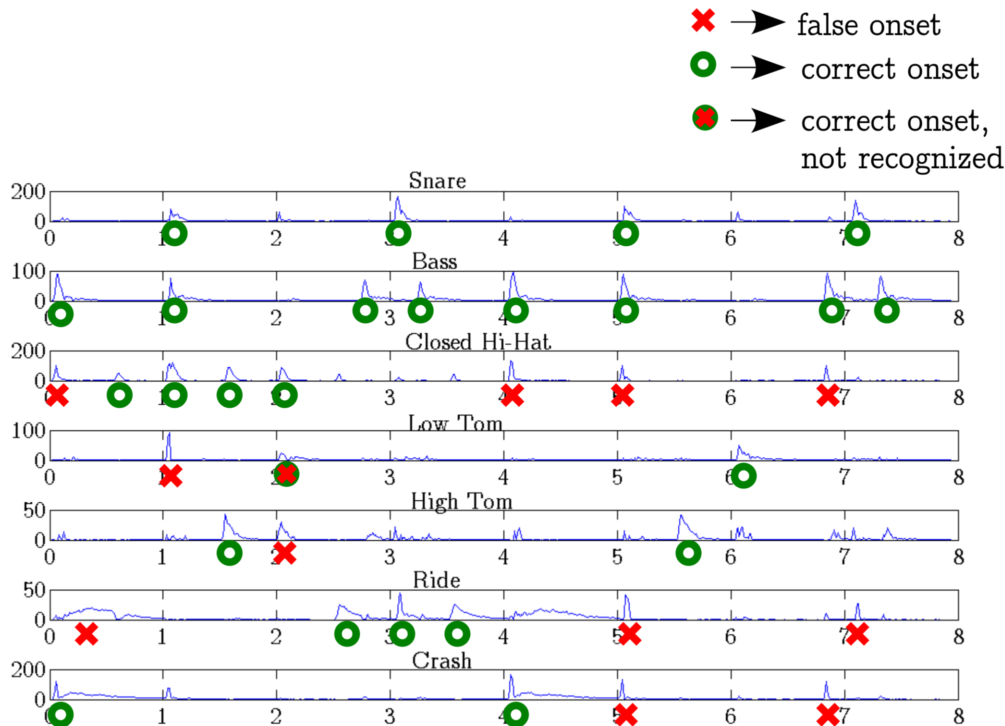


Figure 3.12: Transcription of the seven instruments rhythm (60bpm)

IV

Hardware's design and implementation

4.1 System's overview

The hardware part was implemented for Terasic's DE2-70 development board. DE2-70 hosts one of the biggest FPGAs of Altera's Cyclone II series. It also provides an audio CODEC chipset and a microphone input. The system's overview is illustrated in figure 4.1. It comprises six main blocks, which handle the initialization of the CODEC, the ADC communication, the Hamming window function, the Fourier transform, the magnitude spectrogram computation and the NNMF. Each of the modules is analytically described in the next sections.

The synthesis tool used was Altera's Quartus II 11.0 and simulation of most of the modules was done in Modelsim SE 6.6d. The synthesis resulted to the usage of:

- 18% of the available logic elements (12,309/68,416),
- 31.6% of the available memory bits (364,042/1,152,000),
- 12% of the embedded 9-bit multipliers (36/300), and
- 50% of the available PLLs (2/4).

The implemented system does not include the training stage of the algorithm. However, the training stage is the same with the real-time core that is implemented, extended with the calculations needed in order to find the fixed basis matrices B. These calculations are the supplementary of the real-time core's calculations needed in order to find the gain matrix G, and as such would be implemented in the same way. The implementation of the training stage would require more memory (58% in total, or 667,869/1,152,000, for training samples of length equal to 1.5 seconds). The values of the

fixed basis matrix B are taken from the Matlab's simulation.

The real-time core has a hard real-time requirement of 10ms, since every 441 new samples, that are fetched every (roughly) 10ms, the new time frame's spectrum must be calculated and then approximated by NNMF. For the demonstration purposes 3 LEDs are driven, each corresponding to one of the three sources. Each time a stroke is recognized, the corresponding LED alters its state. A minimum time of 50ms needs to pass for a new stroke on the same source to be recognized. The board's 50MHz clock is the input of both PLLs, which output a 19.93MHz clock needed by the audio CODEC chipset and the internal global clock of the system, equal to 50MHz. The synthesis resulted to a maximum possible frequency of approximately 58MHz. For the debugging needs an UART module is implemented, in order to send to a PC values of various stages of the algorithm. The UART module is taken from [19].

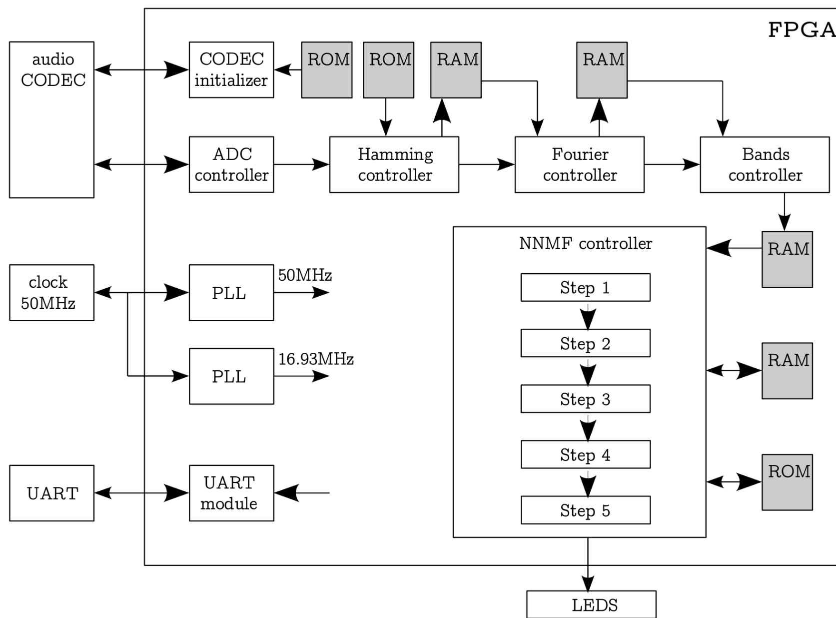


Figure 4.1: The overview of the system

4.2 WM8731 audio CODEC

The WM8731 is an audio CODEC (COder-DECoder) chipset from Wolfson Electronics, part of the development board DE2-70. Its block diagram is shown in figure 4.2. The paths used for the needs of this project are highlighted. Among the other features it hosts, it provides an analog-to-digital converter (ADC), with programmable sample rates in the range 8-96kHz, and word lengths of 16-32bits. It also has a microphone input, with 2 stages of gain made up of two inverting operational amplifiers, allowing microphones of different sensitivities to be used. The first stage comprises a nominal gain of G1=50k/10k=5. By adding an external resistor (R_{mic}) the gain can be adjusted as:

$$G1 = 50\text{k} / (R_{mic} + 10\text{k})$$

DE2-70 uses such a resistor R_{mic}=330Ω, resulting to G1=4.84. The second stage consists of a 0dB gain that can be programmed to provide an additional fixed 20dB.

In order to decide if the second gain stage is needed, the dynamic range of the ADC's outputs was examined. With the help of LEDs, that were flashing whenever the 16-bit signed output of the ADC was greater than $|\pm 4096|, |\pm 2048|$, or $|\pm 1024|$, it was determined that the vast majority of strokes produced values in the range $[|\pm 1024|, |\pm 2048|]$, while more intense strokes were surpassing $|\pm 2048|$, but never $|\pm 4096|$. Hence, the dynamic range of the sampled data is 13bits (sign bit included). If the fixed 20dB gain was used, which concerns a gain equal to 10, 16 bits might not be enough (resulting to unwanted clipping), and therefore it was not used.

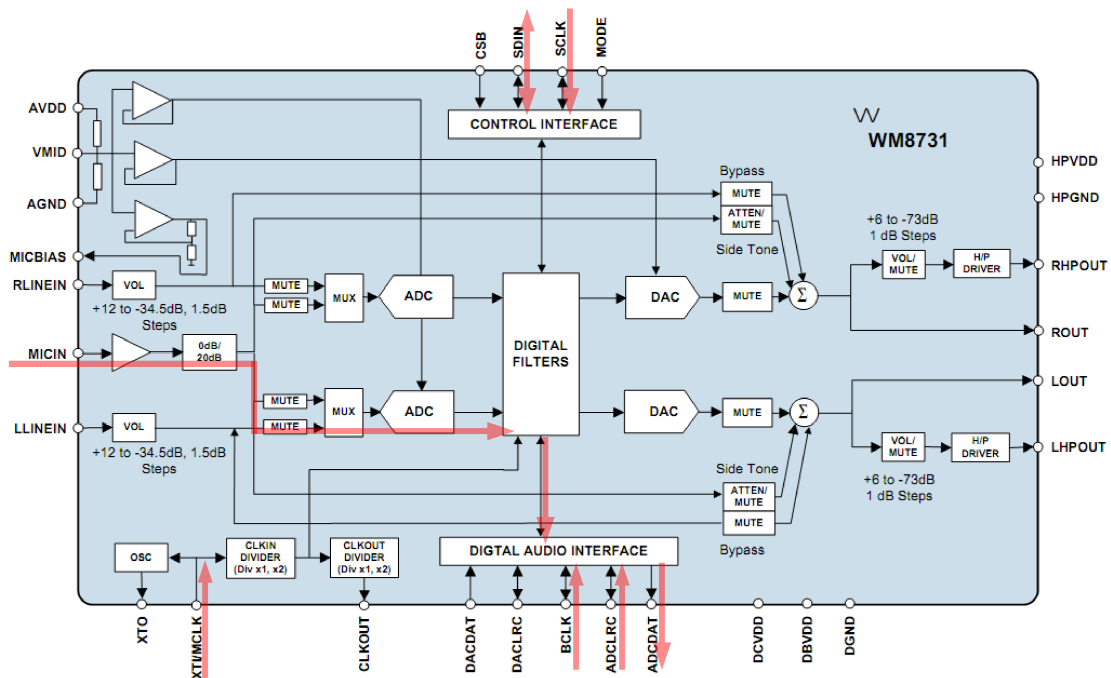


Figure 4.2 (taken from [17]): The block diagram of WM8731

WM8731 can either generate the clock it needs and function as a master device, by connecting an external crystal between the XTI/MCLK input and XTO output pins, or receive its clock by a component other than WM8731 and function as a slave. In the latter case, which is the one used, the external clock is applied directly through the XTI/MCLK input, without any software configuration needed.

In figure 4.3 the interface between WM8731, functioning in slave mode, and the FPGA is outlined. While in slave mode the WM8731 sends the sampled data, ADCDATA, in response to the externally applied clocks, BCLK and ADCLRC. In the next subsection, 4.4.1, the initializer module is described. It configures, over the I^2C, the registers of WM8731 to sample at 44.1kHz, outputting 16 bits words. In 4.4.2 a closer look is taken at how the sampled data are fetched by the ADC controller.

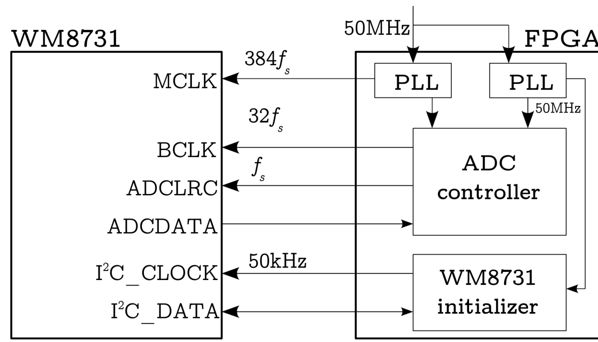


Figure 4.3: The interface between the FPGA and WM8731 in slave mode

4.2.1 Initialization of WM8731

The software control interface of WM8731 let us specify its operating settings. It requires communication on a two-wire serial interface, consisting of the I^2C_CLOCK and I^2C_DATA signals (SCLK and SDIN in the block diagram, respectively). In DE2-70 board's implementation, WM8731 listens only to the address 0011010. The initializer FPGA module initiates a data transfer by establishing a start condition, defined by a high to low transition on I^2C_DATA, while I^2C_CLOCK remains high. This indicates that an address and data transfer will follow. If the correct address is received, and R/W bit is '0', indicating a write, then WM8731 responds by pulling I^2C_DATA low on the next clock pulse (ACK). WM8731 is a write only device and will only respond if R/W is '0'.

Once the correct address has been acknowledged, the initializer sends the first eight data bits (B15-B8, MSB first), WM8731 acknowledges, then the remaining eight bits are sent (B7-B0) and WM8731 acknowledges again. Therefore, 24 bits must be sent for a register to be configured. A stop condition is established with a low to high transition of I^2C_DATA, while I^2C_CLOCK is high. If a start or stop condition is detected out of sequence at any point during the transfer, the device jumps to the idle condition. In case the described sequence of events completes successfully, the WM8731's 9-bit register, with the 7-bit address B15-B9, is updated with the data B8-B0. Figure 4.4 depicts the procedure described above.

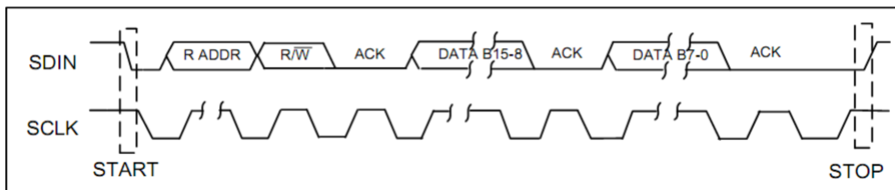


Figure 4.4 (taken from [17]): The two-wire serial interface for the software configuration of WM8731

There are 11 registers in WM8731 and 6 of them need to be configured, while 4 keep their default values and the last one is only used in order to reset the device. Table 4.1 summarizes the addresses of the registers and their values after the configuration.

Register	Address	Register's value	24-bit value (hex) stored in ROM
Left Line In	0	0 1001 0111 (default)	-
Right Line In	1	0 1001 0111 (default)	-
Left Headphone Out	2	0 0111 1001 (default)	-
Right Headphone Out	3	0 0111 1001 (default)	-
Analogue Audio Path Control	4	0 0000 0100	340804
Digital Audio Path Control	5	0 0000 0000	340A00
Power Down Control	6	0 0111 1001	340C79
Digital Audio Interface Format	7	0 0000 0001	340E01
Sampling Control	8	0 0010 0010	341022
Active Control	9	0 0000 0001	341201

Table 4.1: WM8731's register values and addresses

The initializer's block diagram is shown in figure 4.5. Its finite-state machine is illustrated in figure 4.6. An 18bytes (6x24bits) ROM is used to store the registers' values shown in table 4.1. The dataControl signal controls a tri-state buffer, allowing the WM8731 to pull the I^2C_DATA line low, acknowledging that it received 8 bits of data.

A 50kHz clock is generated by a counter, whose input is the main 50MHz clock of our system. I^2C_CLOCK is generated by an OR gate, whose inputs are the counter's 50kHz clock and the FSM's signal clockControl. Any frequency in the range $0<I^2C_CLOCK<526$kHz could be used. When all of the six registers are configured FSM's signal clockControl is kept high, deactivating the software control interface.

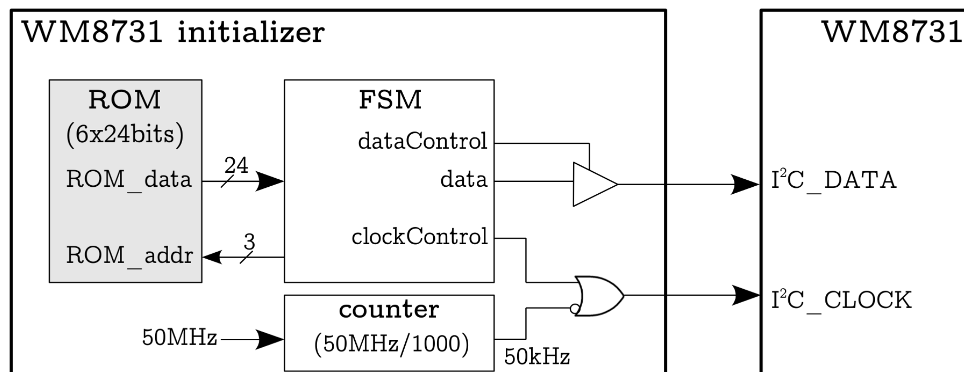


Figure 4.5: The block diagram of the initializer module

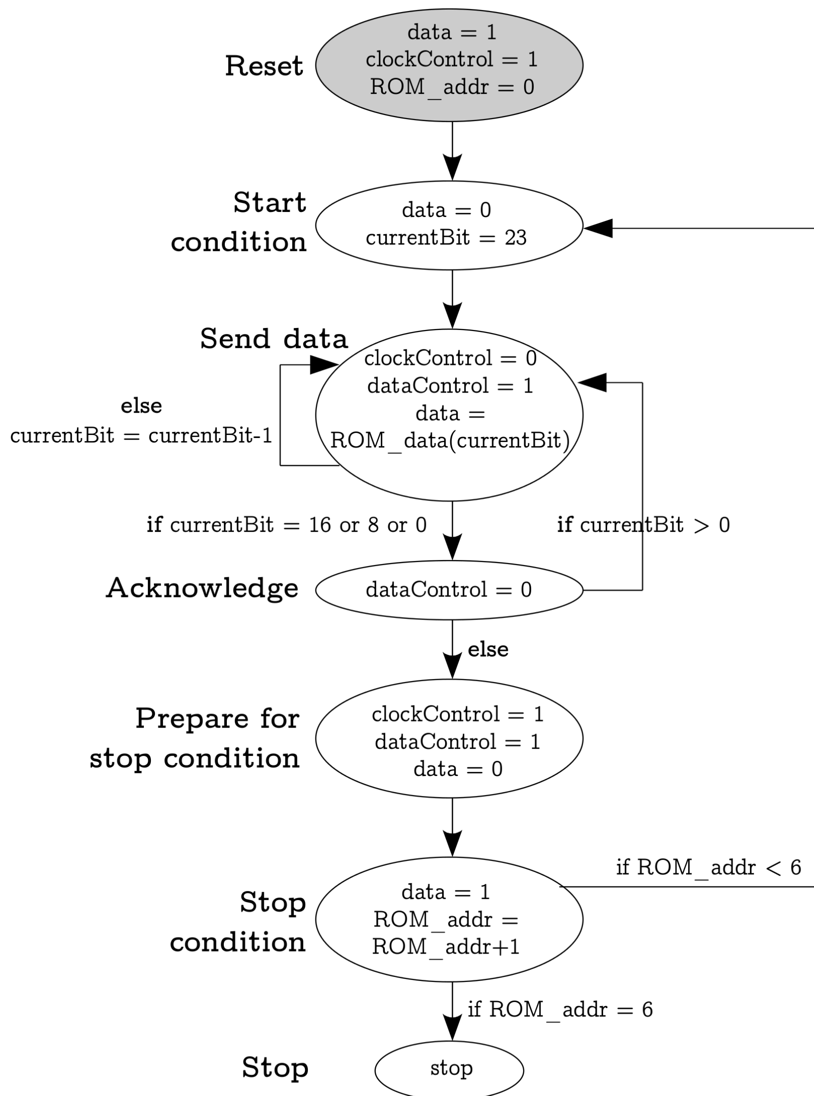


Figure 4.6: The FSM of the initializer module

4.2.2 Fetching the ADC samples

WM8731 can be configured to output the ADC's data in one of the following modes: right justified, left justified, I^2S or the DSP mode. The configured mode in our case is the left justified one, while the length of the output word is equal to 16 bits. In this mode the MSB of the data is available at the first rising edge of BCLK following a ADCLRC transition, as figure 4.7 illustrates. The left and right channels' data are multiplexed. Since in our case ADC's input consists of a single channel, both left and right channels contain the same information.

The 16-bit words are of signed 2's complement format and are being read during the left channel's periods. ADCDATA is synchronous with the BCLK, with each data bit transition signified by a BCLK high to low transition. Each low to high transition of ADCLRC initiates the ADC controller to begin to store the new sample. ADCLRC must

always change on the falling edge of BCLK. The only requirement regarding the frequency of BCLK is to provide sufficient cycles for each ADCLRC transition to clock the chosen data word length (it could even be non-continuous).

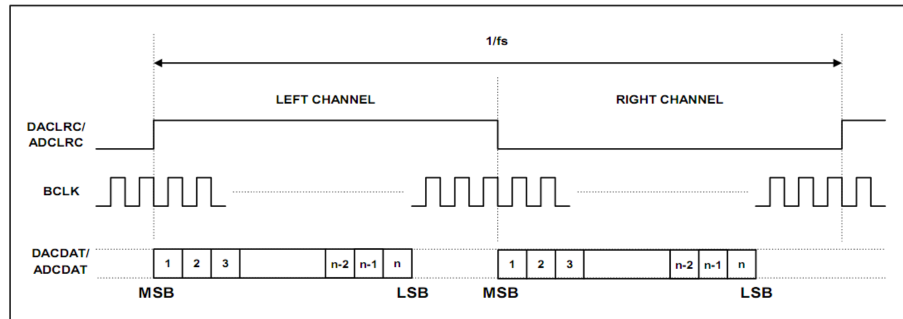


Figure 4.7 (taken from [17]): ADC's output in left justified mode

The chosen sampling rate, f_s, is 44.1kHz and WM8731 is configured to be clocked by MCLK=384f_s. A PLL, whose input is the 50MHz clock, is utilized in order to generate MCLK. Table 4.2 shows the closest value PLL can generate, given the 50MHz input. Our sampling rate is slightly higher than 44.1kHz. For simplicity, the frequency chosen for BCLK is equal to 32f_s, the lowest possible value for data word length of 16 bits. BCLK is generated by a counter, whose input is the MCLK, while ADCLRC is generated by another counter, whose input is the BCLK. The block diagram of the ADC controller is illustrated in figure 4.8.

Clock	Frequency (expected)	Frequency (in practice)
MCLK = $384f_s$	16.9344MHz	16.935484MHz
BCLK = $32f_s$	1.4112MHz	1.41129033MHz
ADCLRC = f_s	44.1kHz	44.102822916kHz

Table 4.2: The approximated frequency values for the three clocks that drive the WM8731

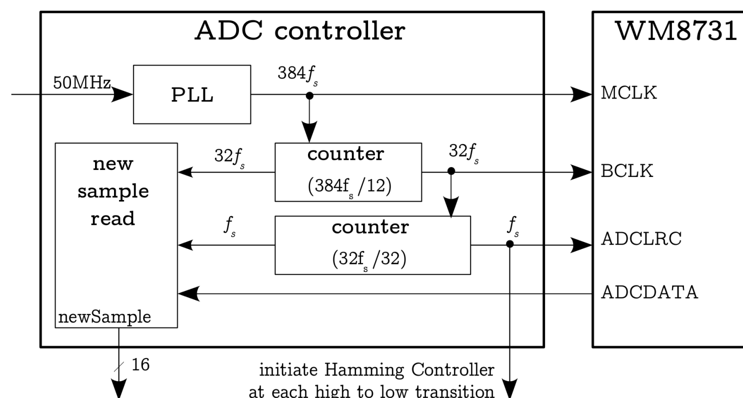


Figure 4.8: The block diagram of the ADC controller module

4.3 Window function

At every high to low transition of ADCLRC, the new sample from ADC is fetched by the Hamming controller. Before it is sent to the FFT module, it needs to be multiplied by the corresponding coefficient of the Hamming window function. Since the hop-size of the STFT is equal to 441 samples and the FFT is applied to 2048 samples, each new sample will take part in either the next four FFT computations, or the next five ones. The Hamming controller is responsible for multiplying each new sample by four or five coefficients and store the results to the hammRAM. Every time 441 new samples are fetched, Hamming controller initiates the next FFT computation, by asserting the signal "enableFFT".

The coefficients of the 2048-point Hamming window function are stored in hammROM. They are approximated by unsigned 8-bit values, in 1.7 fixed-point format, resulting to total size of hammROM equal to 2048bytes. The multiplication of a 16-bit sample (integer) by a sign-extended 9-bit coefficient results to a signed 25-bit product, in 18.7 fixed-point format. Ignoring the 7 fractional bits and the two MSB, beyond the sign, the results are approximated by 16-bit signed natural numbers in 2's complement format. The two most-significant bits, beyond the sign, can be ignored because the coefficients' range is (0,1].

Hamming controller stores the inputs of the five upcoming FFTs in hammRAM, whose size is, therefore, equal to $5 \cdot 2048 \cdot 16\, bits = 20$kB. hammRAM's structure is illustrated in figure 4.9, as well as an example that shows the way Hamming controller stores the values in it. Let an FFT of the 3rd segment (addresses 4096-6143) be the last one computed, and 441-i-1 new samples to have been already fetched. Then, when the (441-i)-th sample arrives, it is firstly multiplied by hammROM[i] and stored to the address 6144+i, since the 4th segment is the next FFT input. Secondly, it is multiplied by hammROM[441+i] and stored to the address 8192+441+i, then to 0+882+i, and so on. If i<2047-1764=283, then the sample will be part of the next five FFTs, and otherwise of only the next four ones.

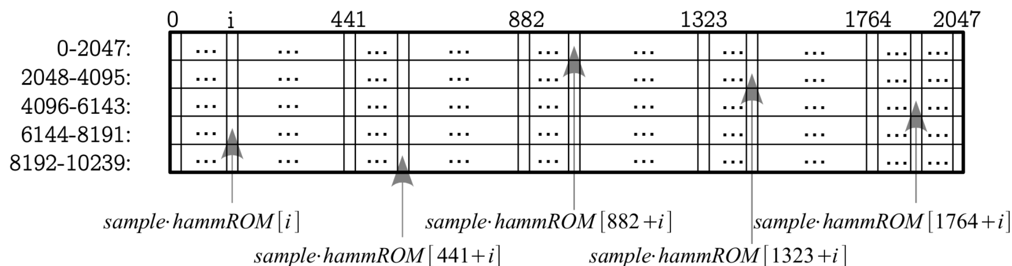


Figure 4.9: hammRAM stores the upcoming five FFT's inputs

The Hamming controller's finite-state machine and block diagram are illustrated in figures 4.10 and 4.11, respectively. It takes 25 cycles for five multiplications with Hamming coefficients to be computed and stored in hammRAM, after each high to low

transition of ADCLRC. However, the next step (Fourier controller) is initiated at the next low to high transition of ADCLRC. Therefore, the latency of Hamming controller is roughly equal to $\frac{1}{2} \cdot \frac{1}{44.1\text{kHz}} \approx 0.011\text{ms}$.

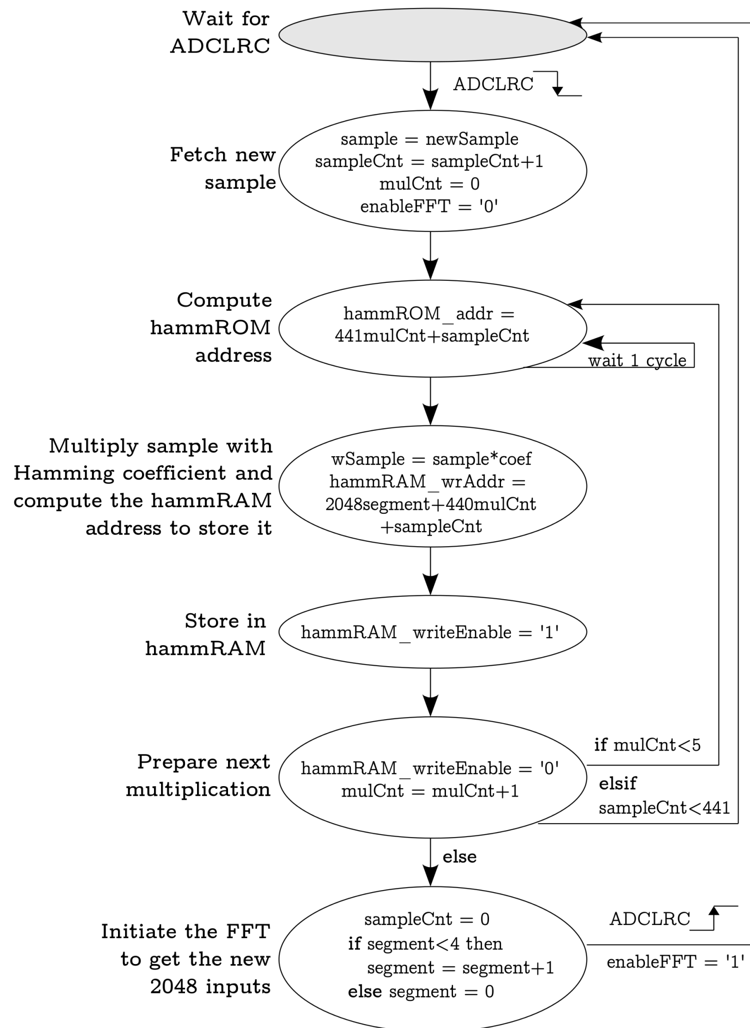


Figure 4.10: Hamming controller's FSM

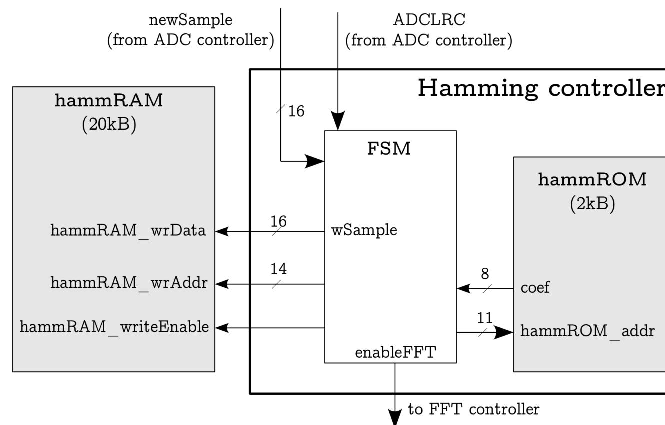


Figure 4.11: Hamming controller's block diagram

4.4 Discrete Fourier Transform

The DFT is based on Altera's IP core ”FFT MegaCore function” ([18]). FFT MegaCore is highly parameterizable, providing architectures for both fixed and variable input lengths. The fixed transform architecture accepts as inputs 2's complement format complex data. In our case the input consists of 2048 16-bit natural numbers, taken by one of the five segments of hammRAM.

FFT MegaCore uses a block-floating-point architecture, which is a trade-off point between fixed-point and full-floating point architectures. Together with the data it also outputs an exponent, which is the same for all 2048 complex values of the output; the output data must be scaled by $2^{-exponent}$ to account for the discarded LSBs during the transform. In case of 2048 input points the exponent is in the range [-16,0]. The parameterization in our case is shown in table 4.3, while the resource usage and cycle count estimation are shown in table 4.4.

Transform length	2048 points
Data precision	16 bits
Twiddle precision	16 bits
FFT engine architecture	Quad output
Number of parallel FFT engines	1
I/O data flow	Burst

Table 4.3: FFT MegaCore function's parameters

Logic elements	3710
Memory bits	114688
9-bit embedded multipliers	24
Transform calculation cycles	2668
Block throughput cycles	6765

Table 4.4: FFT MegaCore function's ressource usage and performance

The burst I/O data flow's interface is illustrated in figure 4.12. It is implemented by the finite-state machines of figure 4.14, part of the Fourier controller, whose block diagram is shown in figure 4.15. The signal sink_ready indicates that the FFT can accept a new block of data. When both sink_ready and sink_valid are asserted the data transfer to FFT occurs. The assertion for one cycle of the signal sink_sop indicates the start of the input block. On the next clock cycle, sink_sop is deasserted and the next 2047 input data samples must be loaded. On the last sample sink_eop must be asserted. The 16-bit wide

sink_real contains our input data, while sink_imag is always equal to zero.

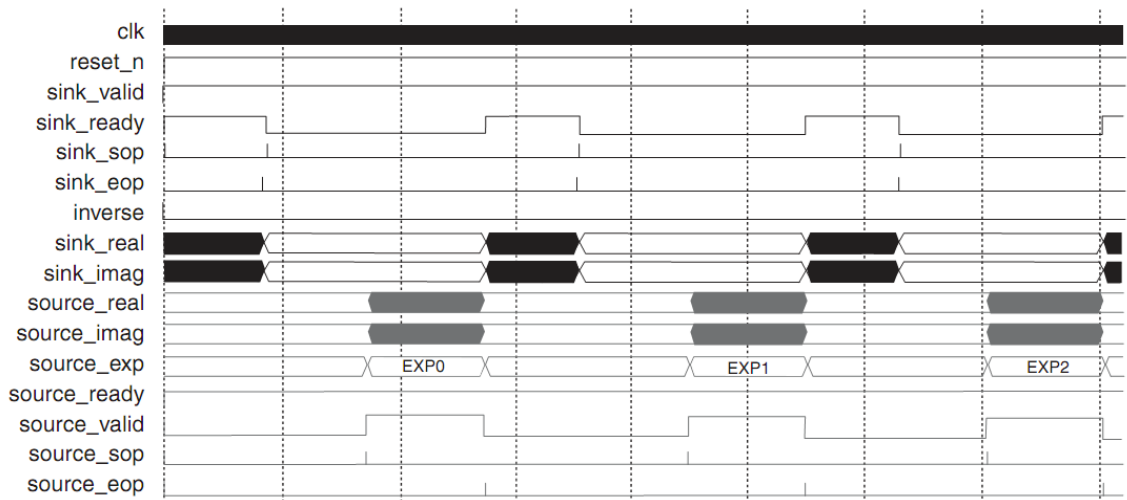


Figure 4.12: The burst I/O data flow interface signals

Once the transform has been completed, FFT asserts source_valid and, if source_ready is asserted, outputs the complex data to the 16-bit source_real and source_imag signals. The exponent of each block is taken from source_exp. The signals source_sop and source_eop indicate the first and last output, respectively. The output data are stored in fourierRAM. We only need to store the first 1024 of them. Since the exponent is in the range [-16,0], each output needs 64 bits, 32 for the real part and 32 for the imaginary. Hence, fourierRAM's size is equal to $64 \cdot 1024$ bits=8kB.

It takes 6765 cycles (see table 4.4) for Fourier controller to read 2048 input data and output the result. But since only half of the outputs are used, the next step is initiated after 6765-1024= 5741 cycles, or roughly 0.115ms for our 50Mhz clock.

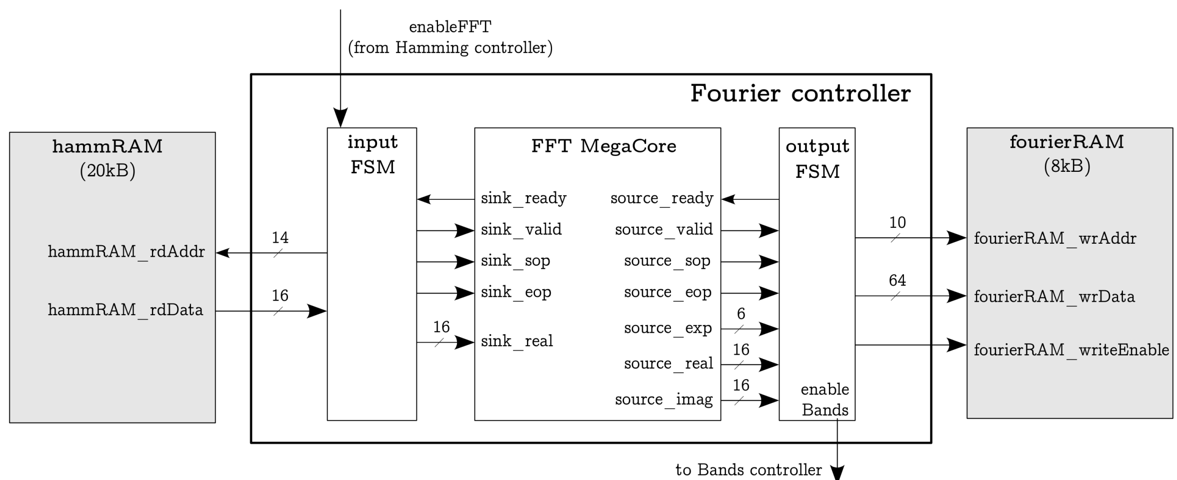


Figure 4.13: Fourier controller's block diagram

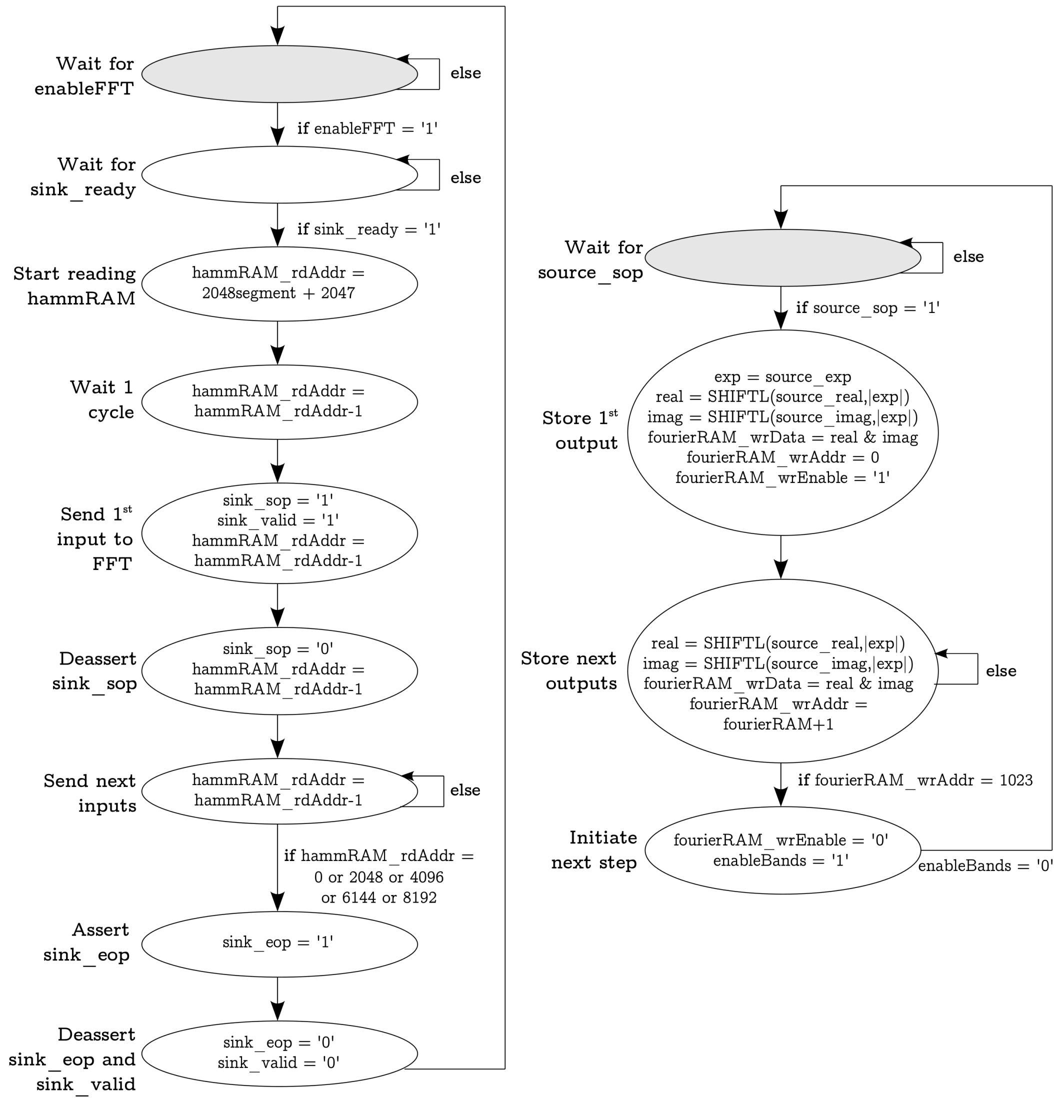


Figure 4.14: Fourier controller's input (left) and output (right) FSMs

4.5 Bandwise magnitude sums

Since the magnitude spectrogram is used, square roots calculations are necessary. If R and I are the real and imaginary parts of a transform's output x, then its magnitude is:

$$|x|=\sqrt{R^2+I^2}$$

The square root's calculation is considered a demanding computationally task, therefore an approximation will just be used. A method which approximates the magnitude of a complex number, called "max plus beta min", is presented in [6]. If MAX=max{|R|, |I|} and MIN=min{|R|, |I|}, then the magnitude approximation is:

$$|x|\approx MAX+b\cdot MIN \quad \text{, where} \quad 0<b\leq 1$$

For simplicity $b=2^{-k}$, where k is a natural number. The average relative errors for b=1, b=0.5 and b=0.25 are shown in table 4.5, based on our test rhythm's data.

Approximation	Average relative error
b=1	27.25%
b=0.5	8.66%
b=0.25	3.19%

Table 4.5: Average relative errors of ”max plus beta min” magnitude's approximation

The Bands controller approximates the magnitude of the 1024 FFT outputs using the ”max plus beta min” with b=0.25. Then, it sums the magnitudes following the 25 critical bands scheme. The block diagram of Bands controller is illustrated in figure 4.15, while its finite-state machine in 4.16. Taking into account the worst-case scenario each magnitude's width must have been 33 bits, while every band's sum 42 bits. However, in order to check if this increase of the number of bits could be ignored, the sums were sent through the UART, so as their range to be determined. As it was expected the widest of the sums, the 25th one, had the maximum value for hi-hat strokes. In case of strokes of normal intensity this was approximately $800,000_{hex}$, while for intense strokes it reached $3,000,000_{hex}$. Hence, they can be represented by 24-bit and 26-bit word lengths, respectively. Therefore, the increase is ignored and the sums' width is equal to 32-bit.

Since the FFT's length is $2048=2^{11}$ points, a scale down by multiplying with 2^{-11} could be applied to the Fourier transform's outputs. Instead of scaling down the 32-bit outputs before the magnitude and sums computations, the scale down occurs after the sums are found. As it was mentioned above the ADC output's dynamic range is 13 bits, while it is represented by 16 bits. In order to account for these 3 bits, a scale down by multiplying with 2^{-8}, instead of 2^{-11}, is implemented. This means that finally the bands sums' widths are 24 bits, resulting to bandsRAM's size of $25 \cdot 24\, bits = 75\, bytes$.

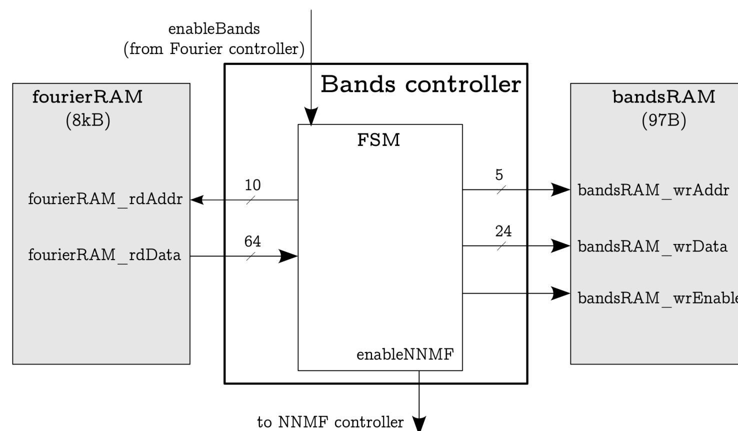


Figure 4.15: Bands controller block diagram

It takes 6 cycles for each FFT output to be read from fourierRAM and its magnitude to be approximated and added to the corresponding band's sum. The first output needs 8 cycles and the storage of the 25 sums to bandsRAM takes 27 cycles. This means that following the assertion of enableBands signal, in total 6173 cycles are needed in order to get each frame's bandwise spectrum in bandsRAM. For a 50MHz clock, the duration of 6173 cycles is roughly equal to 0.123ms.

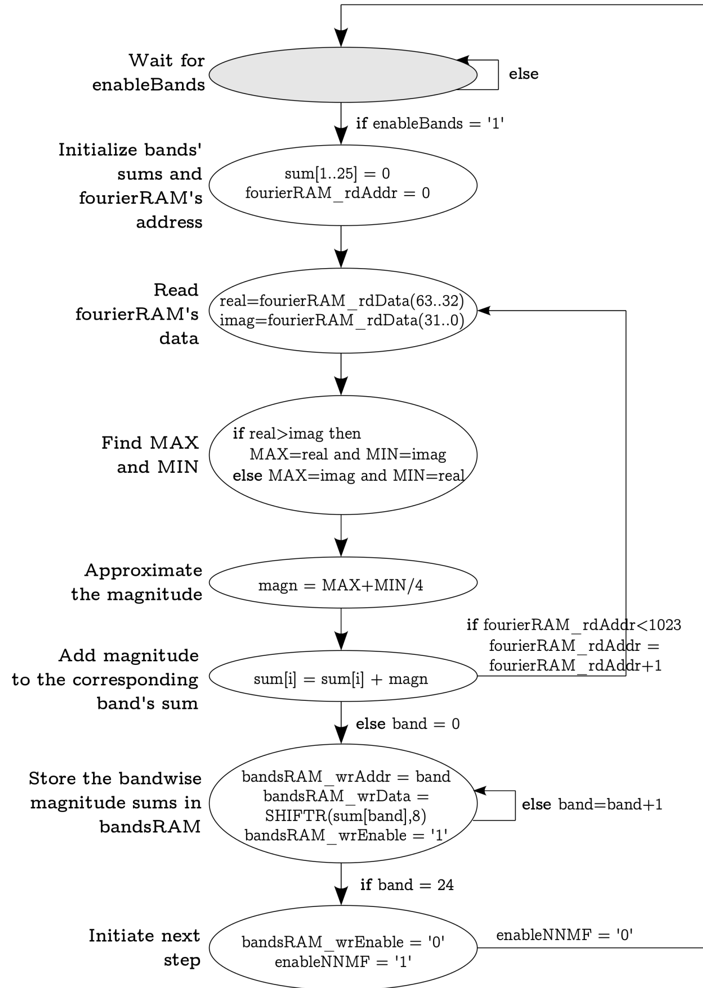


Figure 4.16: Bands controller FSM

4.6 Non-Negative Matrix Factorisation

The NNMF algorithm needs to perform the following calculations, in order to update the value of the gain matrix G and find the new value of the cost function:

$$G^{new} \leftarrow G.\times\left[(B_{\text{fixed}}^{T}\cdot(X./(B_{\text{fixed}}\cdot G)))./(B_{\text{fixed}}^{T}\cdot\mathbf{1}) \right]$$

$$costFunction \leftarrow \sum_{m=1}^{M}\left(X[m]\cdot\log_{10}\left(\frac{X[m]}{(B_{\text{fixed}}\cdot G^{new})[m]}\right) \ - \ X[m] \ + \ (B_{\text{fixed}}\cdot G^{new})[m] \right)$$

The calculations are performed in the following five serially executed steps:

- step 1: $st1 = X./(B_{\text{fixed}} \cdot G)$,
- step 2: $st2 = (B^T_{\text{fixed}} \cdot st1)./(B^T_{\text{fixed}} \cdot \mathbf{1})$,
- step 3: $st3 = G^{new} \leftarrow G. \times st2$,
- step 4 is identical to step 1,
- step 5: $costFunction \leftarrow \sum_{m=1}^{M} \left(X[m] \cdot \log_{10} st1[m] \; - \; X[m] \; + \; (B_{\text{fixed}} \cdot G^{new})[m] \right)$

The three first steps update the matrix G, step 4 re-calculates the matrix $X./(B_{\text{fixed}} \cdot G)$, since its elements' logarithms are needed in step 5, which updates the cost function value. The algorithm iteratively runs the five steps, until the difference of the last two cost function's values is less than a constant threshold.

Fixed-point numbers are used. The widths of the elements of B and G are 12.4 unsigned numbers. The integer part is 12 bits, since the product of B and G approximates the 24-bit unsigned integer spectrum stored in bandsRAM. G's values are initialized to ones, before each spectrum's approximation. Matlab simulation with fixed-point numbers was conducted and according to it, 4 fractional bits results to identical transcription to higher precision. The only impact of the lower precision is the higher divergence threshold needed to be used.

4.6.1 Steps 1-3

The timing diagram of step 1 is illustrated in figure 4.17. Step 1 multiplies B with G and performs the element-wise division $X./(B_{\text{fixed}} \cdot G)$. The matrix BG contains unsigned elements of format 26.8, while the division's results are 16.8 numbers. Every 3 cycles a new element of BG is calculated and a new division starts. The divider is taken from Altera's libraries (lpm_divide). The divident (natural number, stored in bandsRAM) is shifted 16 bits to the left, so as the quotient to have 8 fractional bits. The divider needs a high number of pipeline stages (20 were used), but the divisions' throughput is equal to 3 cycles. The fixed basis matrix B (25x3) is stored in matrixBRAM, while the gain matrix G (3x1) in matrixGRAM. Their sizes are equal to 150 bytes and 6 bytes, respectively. The 25 division's results are stored in RAM_1, whose size is 75 bytes.

Before step 1 proceeds to the calculations, the 3 values of G are initialized to ones. It takes 6 cycles to do so and then 5 more cycles for the first division's divident and divisor to be ready. Then, after 20 cycles the first result is ready, while the 25th one is ready after 72 more cycles. During the following 3 cycles the last result is stored in RAM_1 and step 2 is initiated. Hence, in total step 1 needs 6+5+20+72+3=106 cycles to

complete. Since G's initialization occurs only once at every time frame's spectrum approximation the total number of cycles of step 1 is equal to 106+101(x-1), where x is the total number of NNMF's iterations needed to converge.

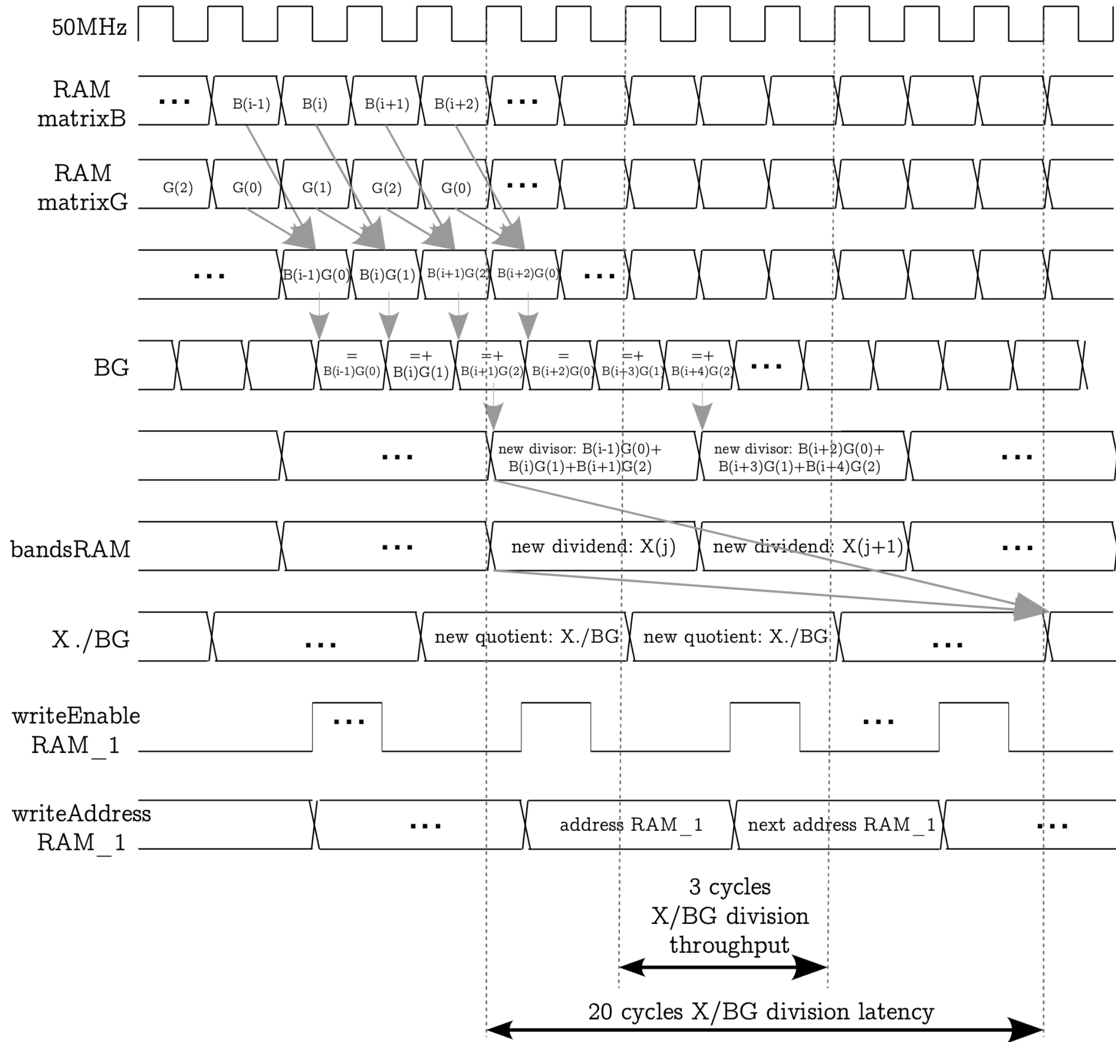


Figure 4.17: Timing diagram of step 1

The timing diagram of step 2 is illustrated in figure 4.18. Step 2 multiplies B^T (3x25) with the result of step 1 (25x1 matrix st1), as well as with a 25x1 matrix which contains only ones. The resulting matrices' elements are 33.12 and 17.4 numbers, respectively. Then, it divides these two matrices element-wise and stores the result in RAM_2. The size of RAM_2 is equal to 9 bytes, since division's results are 16.8 fixed-point numbers. Similarly with step 1, the divider's pipeline depth is 20.

Two cycles after step 2 is initiated, the RAMs values are read and the dividend/divisor values start to be calculated. It takes 25 cycles for a new divident/divisor to be found. In total three divisions are computed. Twenty cycles after the last one, the last result is ready and 2 cycles after, it is stored in RAM_2. Therefore, in total 99x cycles are needed for step 2 to complete, where x is the total number of NNMF's iterations.

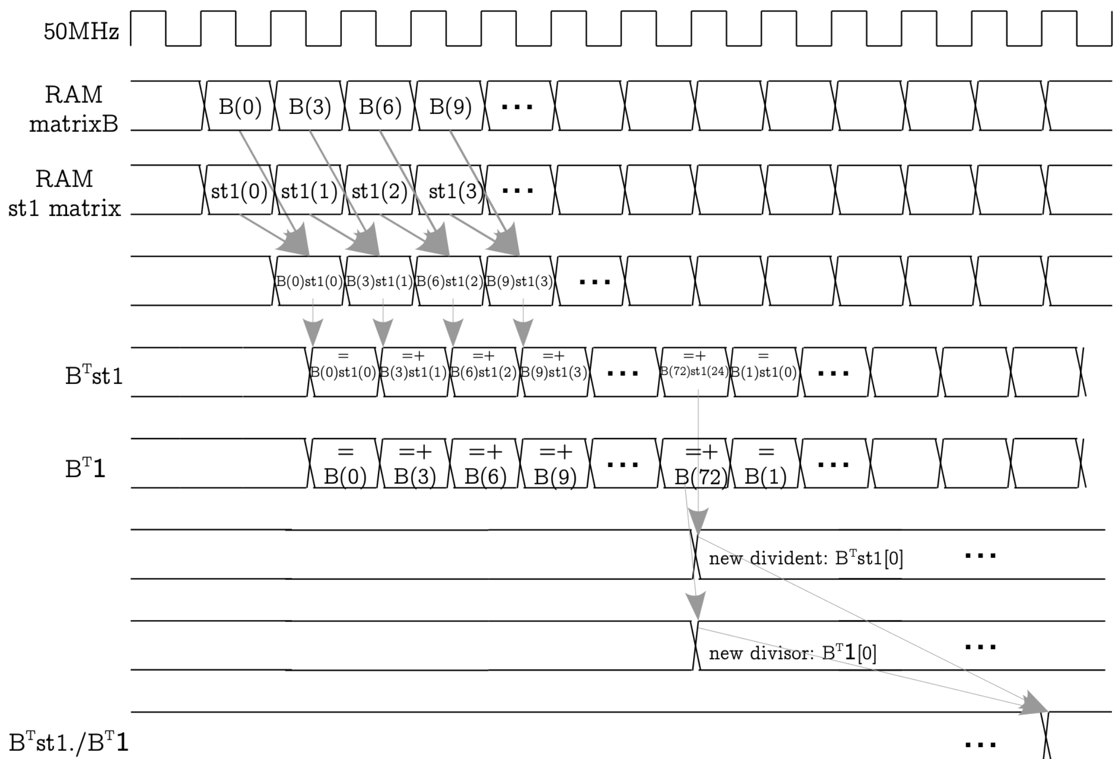


Figure 4.18: Timing diagram of step 2

Step 3 updates matrix G's value, by multiplying element-wise the matrix G with the result of step 2 (3x1 matrix). The timing diagram is illustrated in figure 4.19. Two cycles after step 3 is initiated matrixG's RAM and RAM_2 are read. Five cycles later the three multiplications are computed and step 4 is initiated. Hence, step 3 needs 7x cycles, where x is the total number os NNMF's iterations.

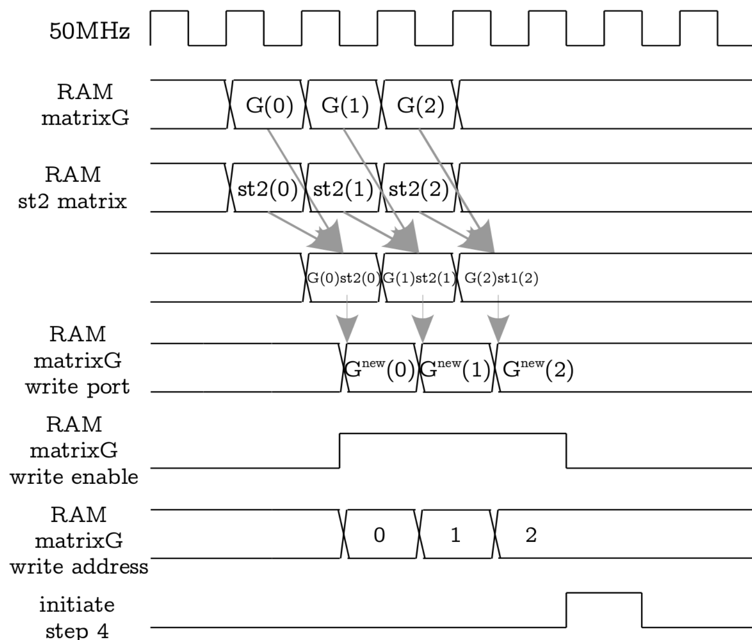


Figure 4.19: Timing diagram of step 3

4.6.2 Steps 4-5

Step 4 is identical to step 1, as it was mentioned above. Step 1 re-executes, needing 101 cycles to complete (since G is never initialized at step 4). Hence, 101x cycles in total are needed for step 4, where x is the total number of NNMF's iterations.

Step 5 comprise the calculation of the logarithms of step 4's results, the multiplication of each logarithm with the corresponding value of bandsRAM, the multiplication of matrices B and G and the additions that give the final cost function's values. The logarithm's value is approximated by linear interpolation, using two look-up tables, logROM and slopeROM. The logarithms of step 4's division results, whose format are 8.8 unsigned numbers, are calculated. Since there is plenty of memory available big look-up tables are used; both logROM and slopeROM's size is equal to 4096 words. Hence, the 12 MSB of the 8.8 number are used as an index for the two ROMs and the 4 LSB as the fraction. logROM stores the pre-calculated values of $\log(y_k)$, where

$$y_k=2^{-8}+k*2^{-4} \text{ and } k=0...4095,$$

with signed 3.14 format. For that precision the mean relative error of the pre-calculated values is equal to 0.00971%, while it would be roughly equal to 3% for a 3.9 format. The slopeROM stores the pre-calculated slope values:

$$\text{slope}(k)=[\log(y_{k+1})-\log(y_k)]/2^{-4},$$

with unsigned (since logarithm is an increasing function) format of 5.14. The logarithm's approximation is given by:

$$\log(X[m]/(BG)[m]) = \text{logROM}[\text{addr}] + \text{slopeROM}[\text{addr}]*\text{fraction}$$

where addr is equal to the 12 MSB bits, while fraction equals the 4 LSB bits.

In figure 4.20 the timing diagram of step 5 is illustrated. Eight cycles after it is initiated, the first cost function's element is computed ($X(0)*\log(0) - X(0) + BG(0)$). It takes 7 cycles for each of the next 24 ones to be computed. Their sum is the cost function's value. In total 183x cycles are needed for step 5 to complete, where x is equal to NNMF's iterations. Finally, 5 cycles after the last iteration, the values of G have been compared to the corresponding thresholds and in case they are greater than them, the signals to LEDs are updated.

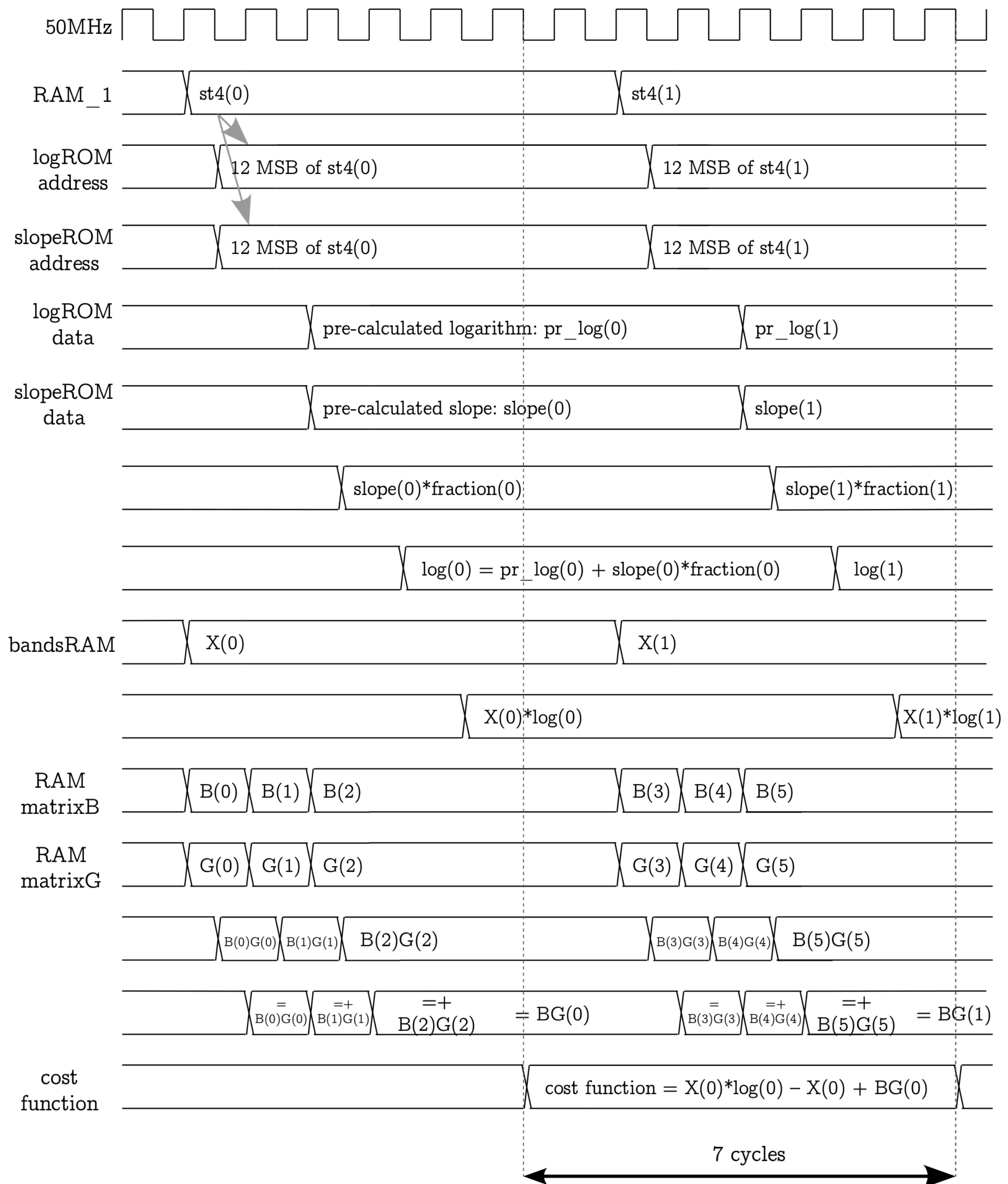


Figure 4.20: Timing diagram of step 3

4.7 Latency and onset time detection

The latency since the 441st sample is read from ADC, the last one Fourier transform waits for, until the LEDs alter their state, in case of a recognized stroke, is summarized below:

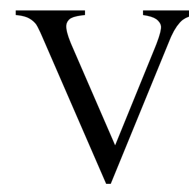
- ADC controller → Hamming controller: there is no latency, since Hamming

controller fetches the new sample at ADCLRC's high to low transitions. In order to fetch the value earlier, a higher frequency for BCLK must be used.

- Hamming controller → Fourier controller: the latency in our case is equal to half of ADCLRC's period (approximately 0.011ms), although it could be 25 cycles (0.5us for our 50MHz).
- Fourier controller → Bands controller: 5741 cycles, or approximately 0.115ms.
- Bands controller → NNMF: 6173 cycles, or approximately 0.123ms.
- NNMF → LEDs: in total the five steps' latency is equal to 491x+10 cycles, where x is the number of iterations NNMF needed to converge.

In case 1000 iterations were needed for NNMF to converge, the NNMF stage would need roughly 10ms in order to complete. However, the algorithm in simulation showed that such a high number of iterations is unnecessary. In fact, as it was shown in 3.4.4 any average number of iterations greater than 10 gives identical transcription results. Matlab's simulation for the fixed-point numbers, showed us that in order to get an average number of iterations equal to 100, a divergence threshold equal to 200 must be used. In addition, ignoring the cost function's value and forcing always ten iterations to occur, had no impact on the performance, too. The latter means that 4920 cycles could be enough for NNMF, resulting to total latency of 0.011+0.115+0.123+0.0984=0.3474ms.

The latency just described concerns only the computational part. In order to get an idea of what the latency of the onset's detection could be, other factors should be also taken into account. In an ideal case, let a single sample turn a time frame with no recognized strokes, into one with at least one recognized stroke. This sample may come first after a new Fourier computation has just began. Then, there is an additional latency of roughly 10ms, until the next transform starts. And beyond that, the fact that window functions are used must be also taken into account. Because the last 441 samples of the 2048-points FFT are always multiplied by the lowest values of the window function (less than 0.4), and possibly an onset will be cancelled for (at least) one transform. Summing the above factors, an estimation that an onset is signalled from LEDs approximately 10-40ms after it occurs seems logical.



Conclusion

Regarding the hardware implementation, the performance of the system meets the expected one, since the vast majority of strokes are correctly recognized. Unrecognized strokes are more frequent in case of simultaneous strokes on at least two instruments. The tests were performed on the same drum kit, using the simulation's fixed basis matrix of the same kit's recordings.

Regarding the specific NNMF-based approach, it has been shown, in both real's and simulation's results, that it behaves very reliably. In case more than one components per source are used, the transcription is reliable even with more instruments present. It could form a very computationally effective, automatic transcription system, together with an algorithm that combines the various components' information, in order to extract the correct onsets. However, even the minimal core that was implemented could be used, without further additions, to implement a tempo tracker system; that is a system which focuses only on snare and bass drums, in order to extract the variations in tempo during a drumming performance.

Evaluating the work on the thesis project in general, very useful experience was gained on VHDL programming, a main thesis project's motivation. In addition, approaching the automatic transcription problem from scratch, the literature survey, and building an FPGA "friendly" system has been an invaluable, instructive experience. A very important lesson taught regards to the debugging procedure of the VHDL code. Surprisingly, at some premature tests the system worked for the first time, although bugs were present in various stages of the data flow. The performance was very poor, barely half of the strokes were recognized, but the fact that it showed that it is working was misleading itself. In the sense that this poor performance was pointing to possible bugs at some fixed-point calculations, where errors in following the radix point, or manipulating the data word widths, may easily lead to not so critically erroneous behavior. Indeed there

were such bugs, whose corrections though had the opposite effect to system's performance. Until a bug at the very first stage of the algorithm was found and, as always, regarded the simplest thing; that was a wrong configuration bit for the CODEC's registers, that resulted to a wrong frequency in the clocks of ADC controller.

References

[1] Klapuri, Anssi, ”Introduction to Music Transcription”, in ”Signal Processing Methods for Music Transcription”, 2006

[2] FitzGerald, Gerry and Paulus, Jouni, ”Unpitched Percussion Transcription”, in ”Signal Processing Methods for Music Transcription”, 2006

[3] Klapuri, Anssi "Sound onset detection by applying psychoacoustic knowledge," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, Phoenix, Arizona, USA, 1999.

[4] J. Vos and R. Rasch, "The perceptual onset of musical tones," *Perception and Psychophysics*, vol. 29, no. 4, pp. 323-335, 1981.

[5] Simon Dixon, "Onset Detection Revisited," in *Proc. DAFx-06*, pp. 133-137, 2006.

[6] http://www.dsprelated.com/showmessage/38751/1.php

[7] Peter Soderquist and Miriam Leeser, "Division and Square Root: Choosing the Right Implementation," *IEEE Micro*, Vol.17 No.4, pp.56–66, July/August 1997

[8] Smith, J.O. "*Spectral Audio Signal Processing,*" http://ccrma.stanford.edu/~jos/sasp/, online book.

[9] “Windowing: Optimizing FFTs Using Window Functions”, http://zone.ni.com/devzone/cda/tut/p/id/4844, National Instruments

[10] http://ccrma.stanford.edu/planetccrma/software/

[11] Barry Truax, http://www.sfu.ca/sonic-studio/handbook/Mel.html, “Handbook for Acoustic Ecology”, 1999

[12] Paulus, Jouni and Virtanen, Tuomas, “Drum Transcription with Non-Negative Spectrogram Factorisation”, 2005

[13] Virtanen, Tuomas, ”Unsupervised learning methods for Source Separation in Monaural Music Signals”, in ”Signal Processing Methods for Music Transcription”, 2006

[14] Daniel Lee and Sebastian Seung, “Algorithms for Non-negative Matrix Factorization”, 2001

[15] Michael Berry and Murray Browne, “Algorithms and Applications for Approximate Nonnegative Matrix Factorization”, 2006

[16] Julius O. Smith III, “Bark and ERB Bilinear Transforms”,
https://ccrma.stanford.edu/~jos/bbt/bbt.html

[17] Wolfson's WM8731 datasheet,
http://www.wolfsonmicro.com/documents/uploads/data_sheets/en/WM8731.pdf

[18] FFT MegaCore Function User Guide,
http://www.altera.com/literature/ug/ug_fft.pdf

[19] Advanced Computer Architecture (DTU 02211),
http://www2.imm.dtu.dk/courses/02211/ ,
http://en.wikiversity.org/wiki/Computer_Architecture_Lab

Appendix A

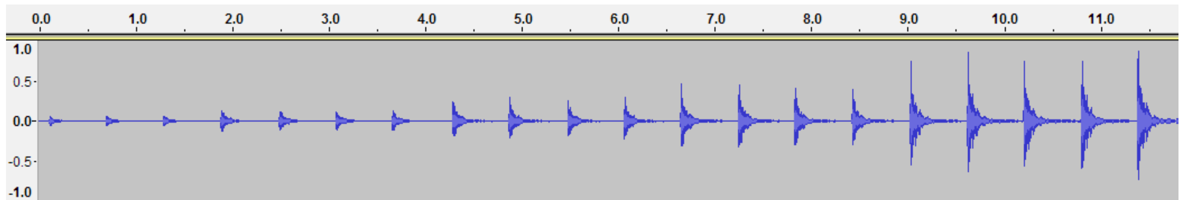


Figure A.1: Snare's dynamics (recording)

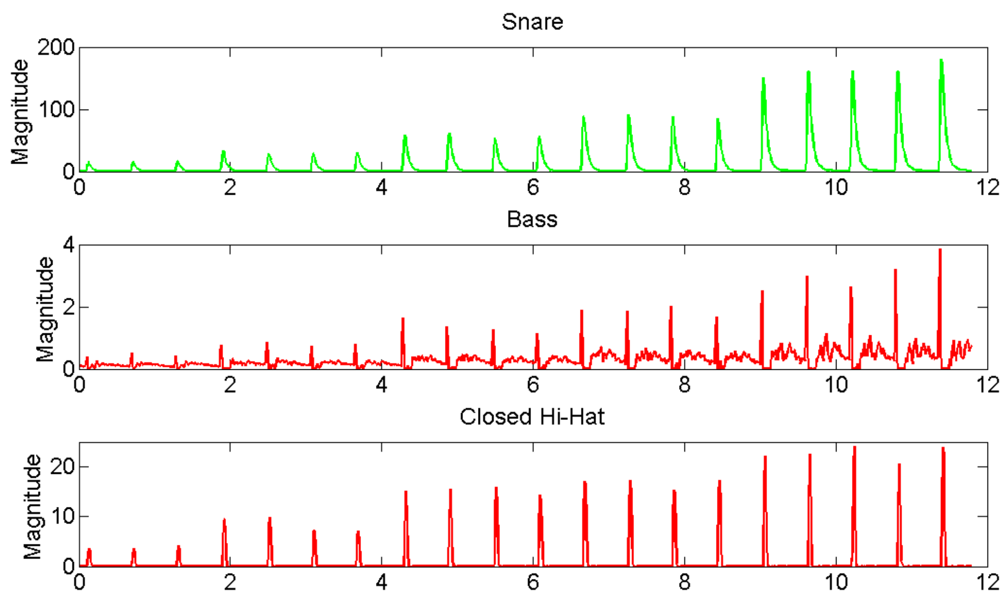


Figure A.2: Transcription of snare's dynamics (considerable noise on hi-hat)

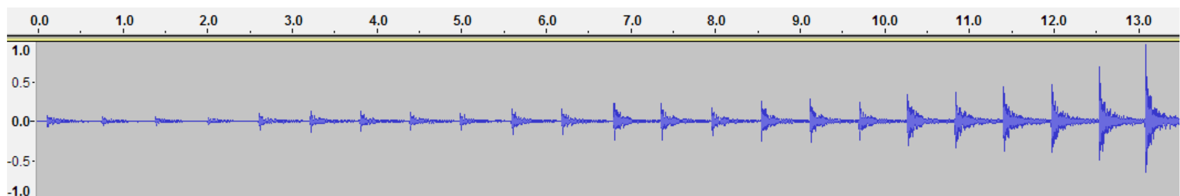


Figure A.3: Basse's dynamics (recording)

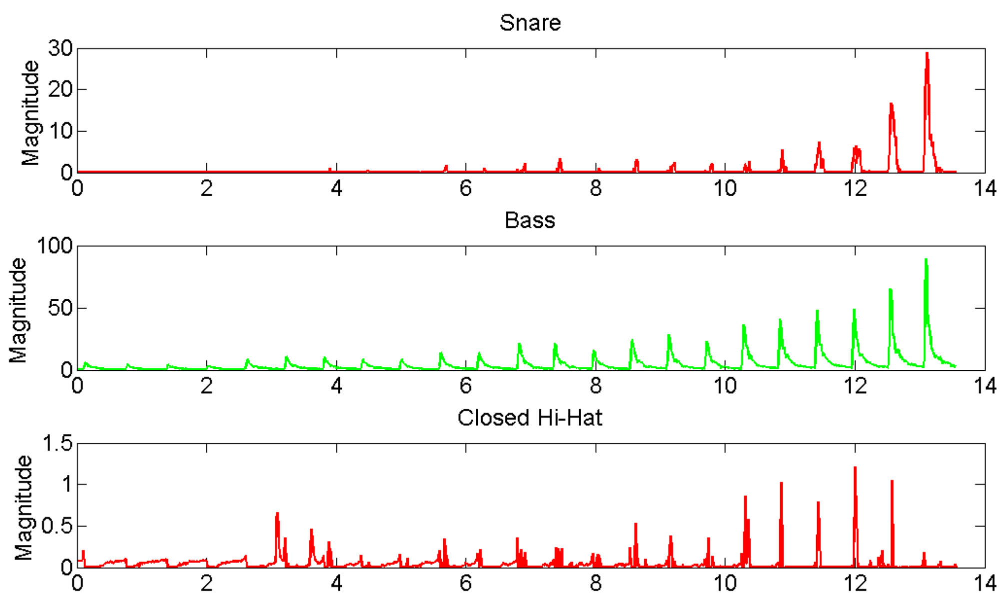


Figure A.4: Transcription of basse's dynamics (considerable noise on snare, for very intense strokes)

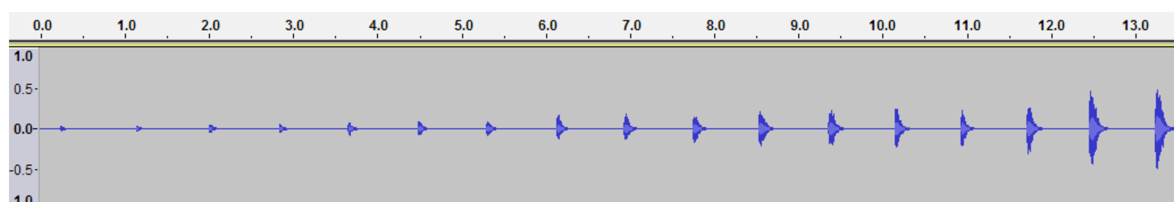


Figure A.5: Hi-hat's dynamics (recording)

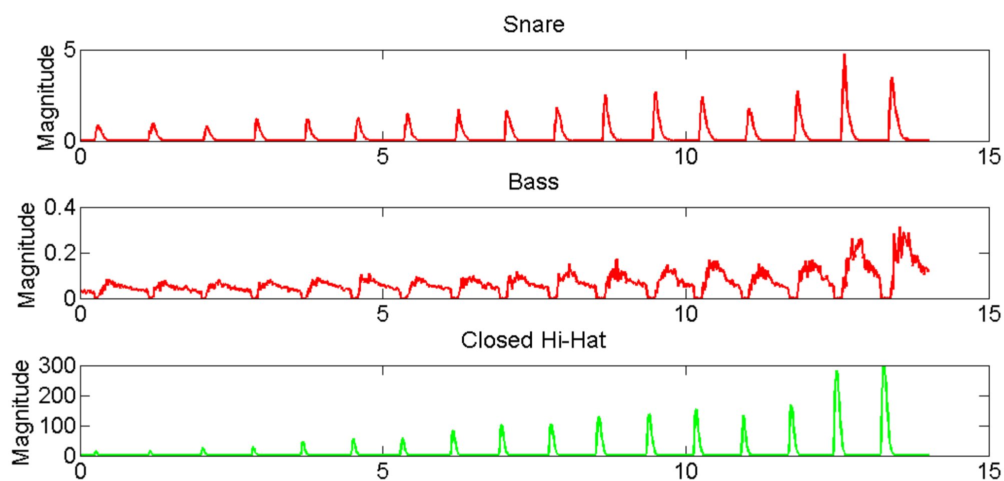


Figure A.6: Transcription of hi-hat's dynamics (negligible noise on both snare and bass)

Appendix B

Table B.1 shows the B_{snare} basis matrix values, calculated for three different values of NNMF's divergence thresholds (10^{-11}, 10^{-2}, 100).

frequency band (Hz)	B_{snare} for divergence threshold $= 10^{-11}$	B_{snare} for divergence threshold $= 10^{-2}$	B_{snare} for divergence threshold $= 100$
0-100	1.3726	1.1095	1.2348
100-200	7.8376	6.3353	7.0508
200-300	28.1783	22.7770	25.3495
300-400	11.6785	9.4399	10.5061
400-510	11.0821	8.9579	9.9696
510-630	12.2025	9.8635	10.9775
630-770	10.1586	8.2114	9.1388
770-920	11.1325	8.9985	10.0148
920-1080	6.3861	5.1620	5.7450
1080-1270	5.6378	4.5571	5.0718
1270-1480	6.0282	4.8727	5.4230
1480-1720	4.9896	4.0332	4.4887
1720-2000	4.6511	3.7595	4.1841
2000-2320	6.0283	4.8728	5.4231
2320-2700	5.2454	4.2400	4.7188
2700-3150	6.9593	5.6253	6.2606
3150-3700	6.9419	5.6113	6.2450
3700-4400	7.6547	6.1874	6.8862
4400-5300	9.5331	7.7057	8.5760
5300-6400	10.3215	8.3430	9.2853
6400-7700	11.4783	9.2781	10.3260
7700-9500	19.6498	15.8832	17.6771
9500-12000	19.3563	15.6460	17.4130
12000-15500	13.9664	11.2893	12.5643
15500-22050	11.1513	9.0137	10.0318

Table B.1: Snare's fixed basis matrices for three divergence thresholds

Appendix C

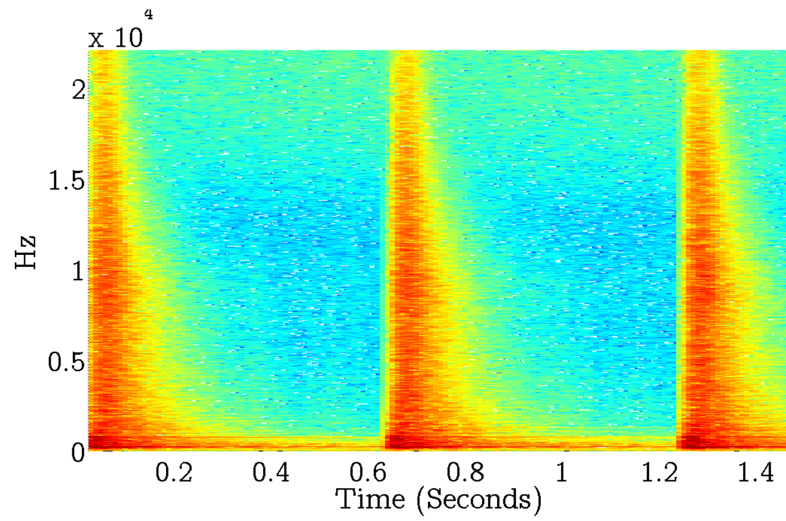


Figure C.1: Snare's training sample spectrogram

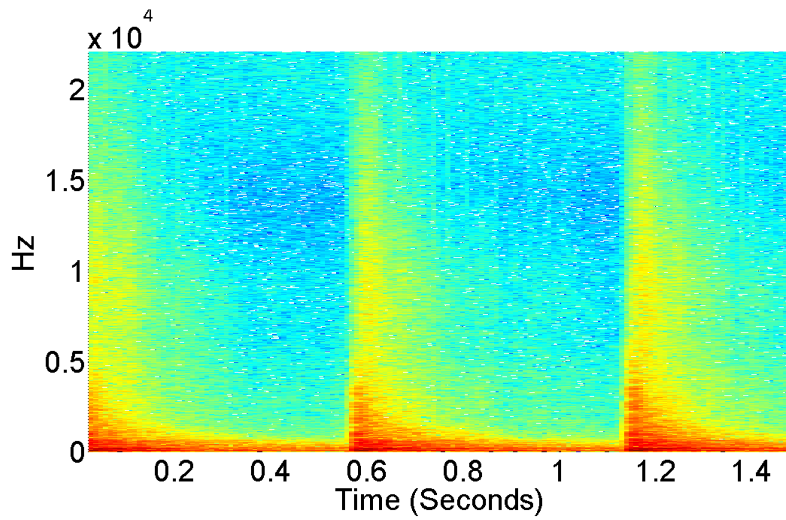


Figure C.2: Bass' training sample spectrogram

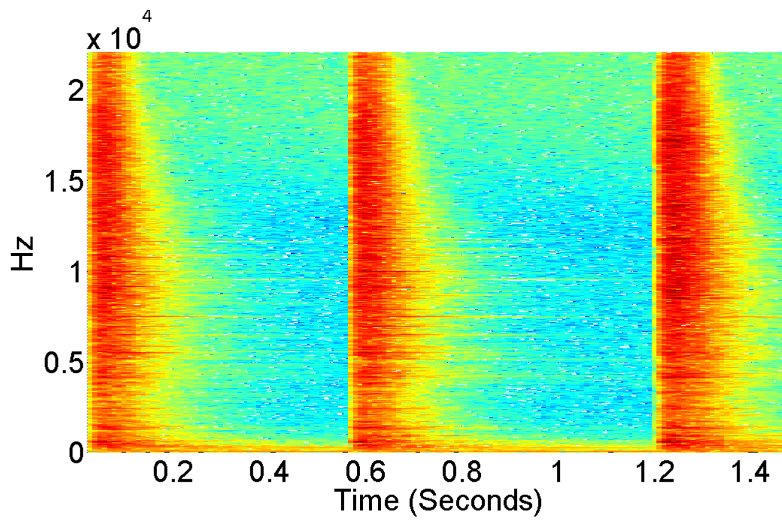


Figure C.3: Closed hi-hat's training sample spectrogram

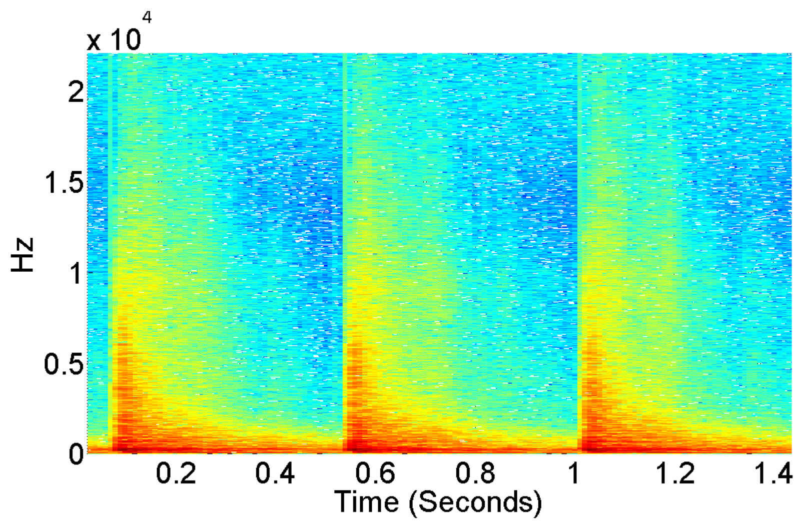


Figure C.4: Low tom's training sample spectrogram

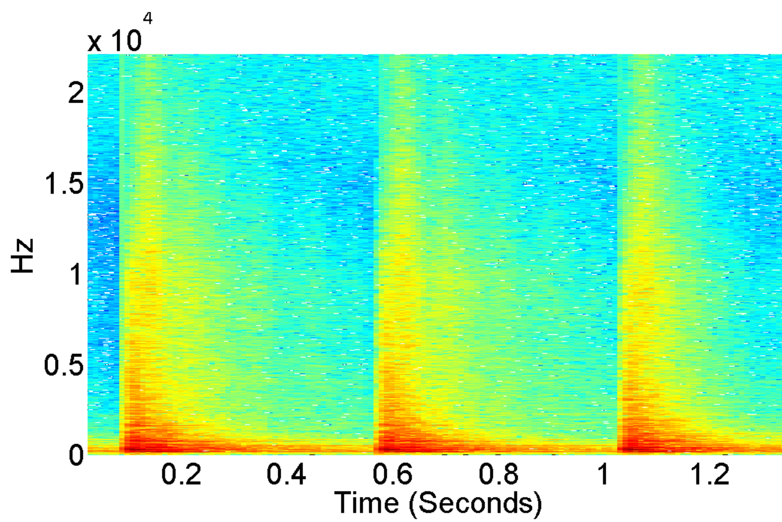


Figure C.5: High tom's training sample spectrogram

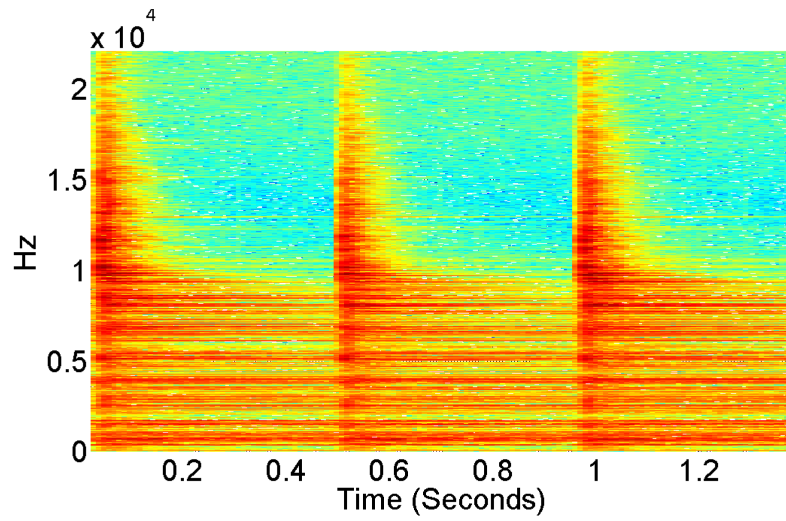


Figure C.6: Ride's training sample spectrogram

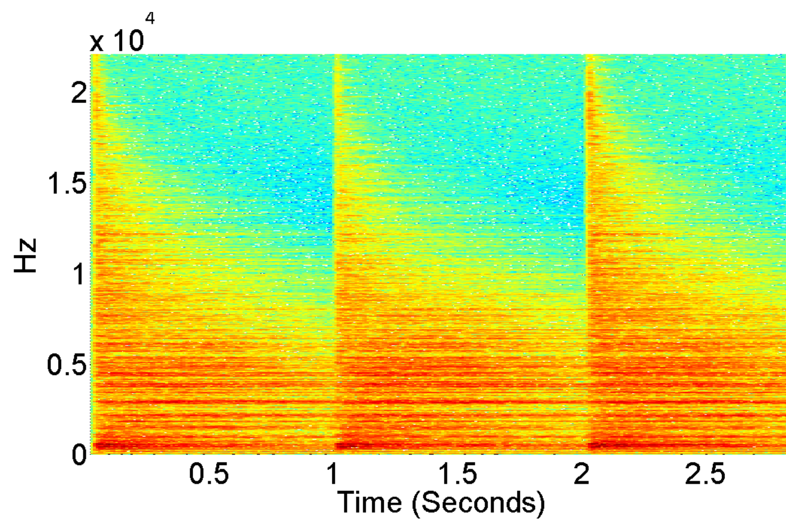


Figure C.7: Crashe's training sample spectrogram

Appendix D

The transcription of the seven-instruments 60bpm rhythm follows. Each source is represented by seven components; the useful ones are highlighted. In all the cases, except the hi-hat's, by adding the highlighted components of each source, the recognition results to 100% success rate. The hi-hat could be recognized correctly 3 out of 4 times, but also 2 false onsets are recognized.

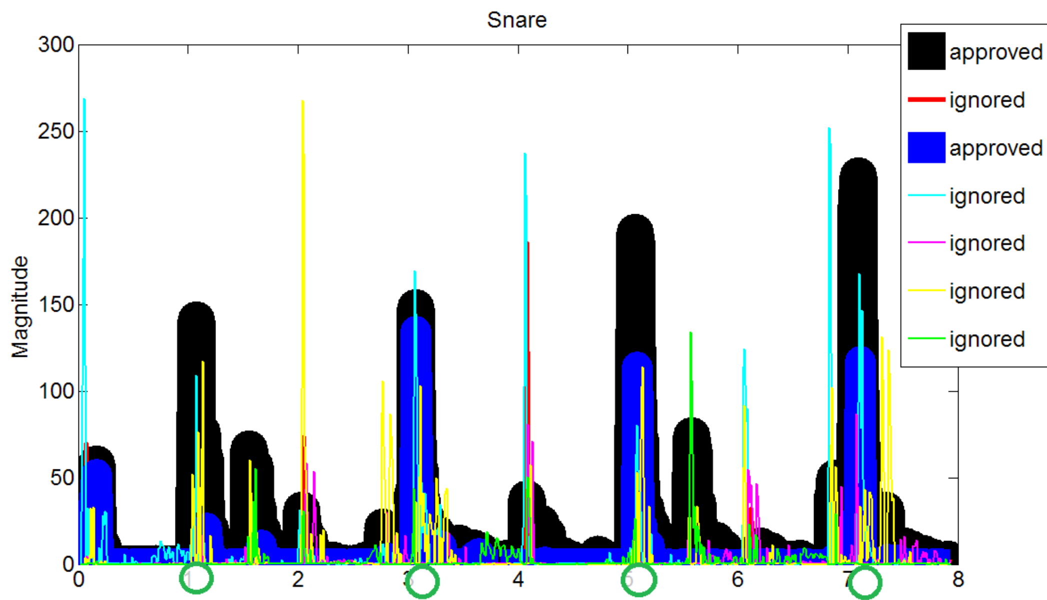


Figure D.1: Transcription of snare for the seven-instruments rhythm

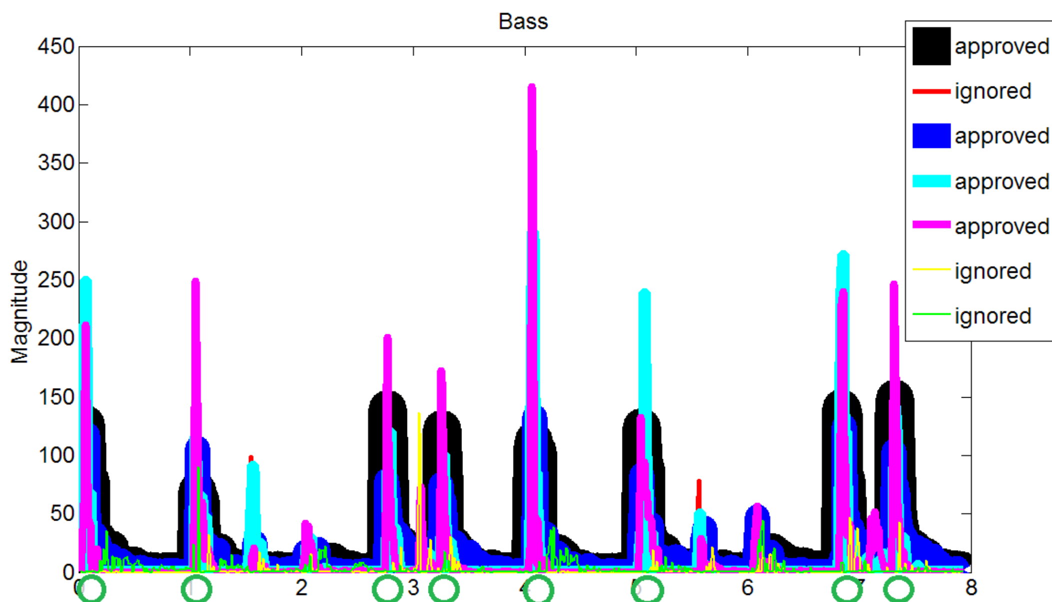


Figure D.2: Transcription of bass for the seven-instruments rhythm

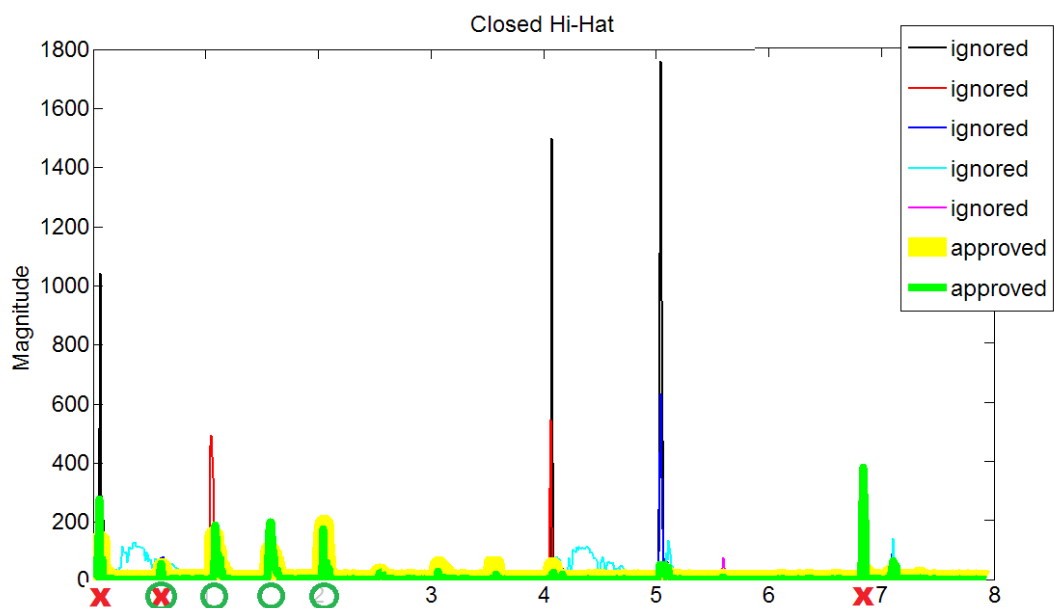


Figure D.3: Transcription of closed hi-hat for the seven-instruments rhythm

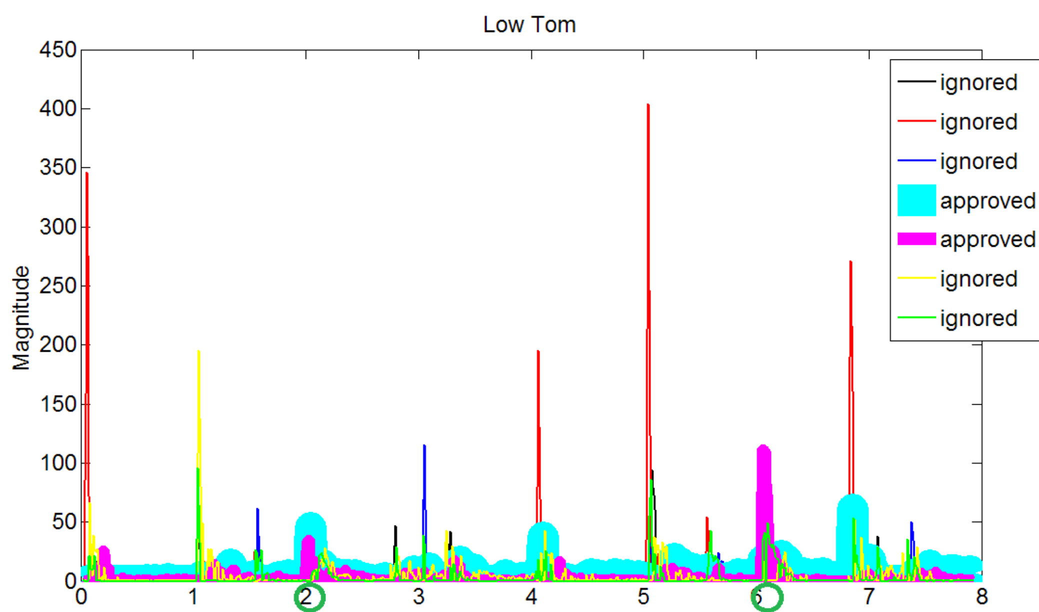


Figure D.4: Transcription of low tom for the seven-instruments rhythm

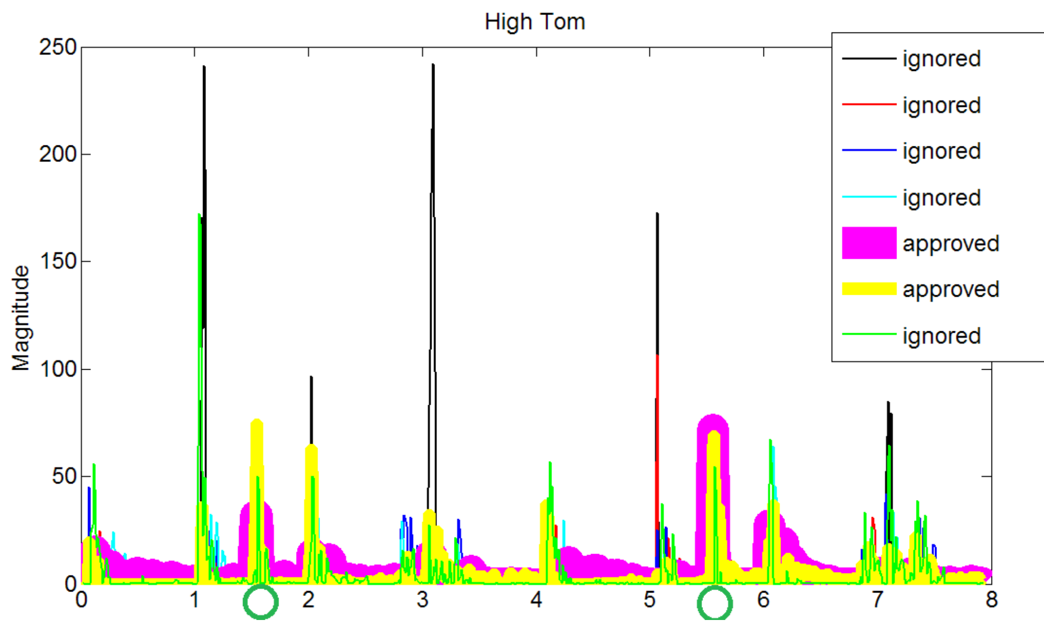


Figure D.5: Transcription of high tom for the seven-instruments rhythm

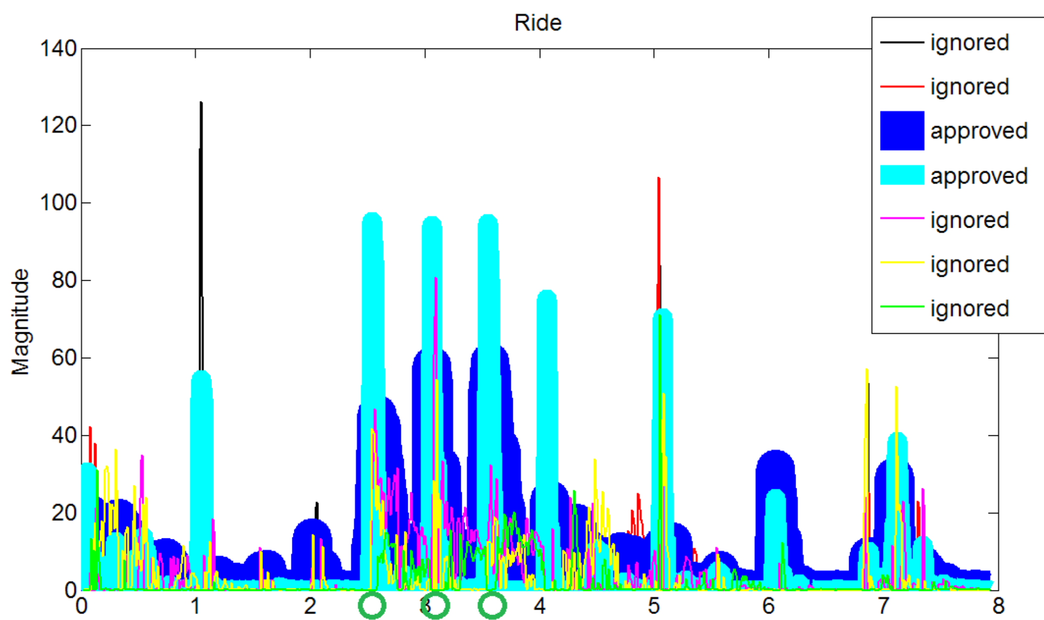


Figure D.6: Transcription of ride for the seven-instruments rhythm

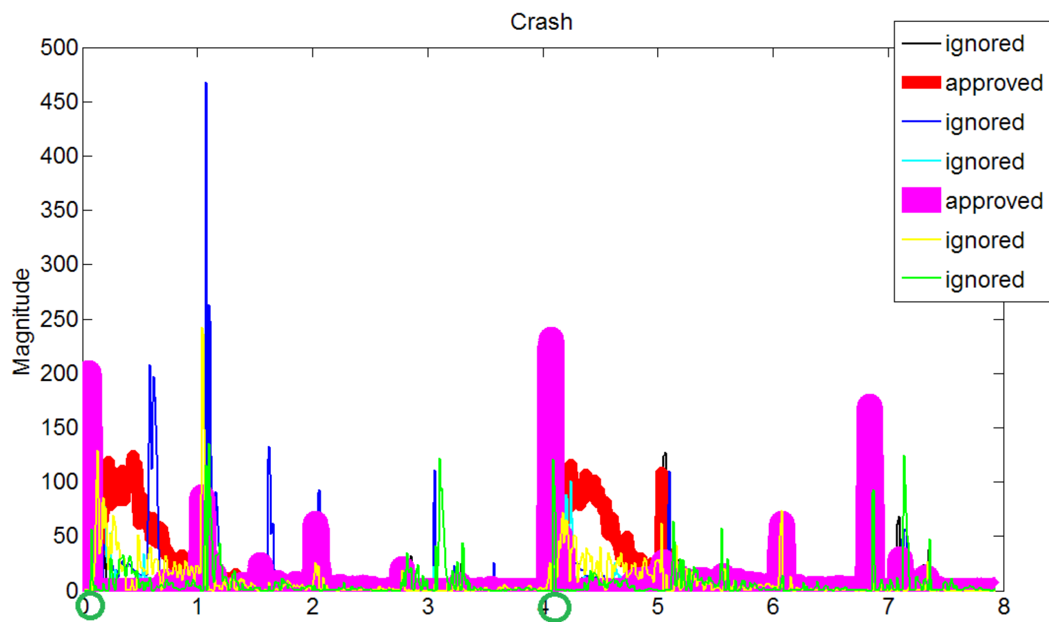


Figure D.7: Transcription of crash for the seven-instruments rhythm