Original Software Publication (OSP)

Title.

NAVIS: Neuromorphic Auditory VISualizer Tool

Authors.

Juan P. Dominguez-Morales, A. Jimenez-Fernandez, M. Dominguez-Morales, G. Jimenez-Moreno

Abstract.

This software presents diverse utilities to perform the first post-processing layer taking the neuromorphic auditory sensors (NAS) information. The used NAS implements in FPGA a cascade filters architecture, imitating the behavior of the basilar membrane and inner hair cells and working with the sound information decomposed into its frequency components as spike streams. The well-known neuromorphic hardware interface Address-Event-Representation (AER) is used to propagate auditory information out of the NAS, emulating the auditory vestibular nerve. Using the information packetized into aedat files, which are generated through the jAER software plus an AER to USB computer interface, NAVIS implements a set of graphs that allows to represent the auditory information as cochleograms, histograms, sonograms, etc. It can also split the auditory information into different sets depending on the activity level of the spike streams. The main contribution of this software tool is that it allows complex audio post-processing treatments and representations, which is a novelty for spike-based systems in the neuromorphic community and it will help neuromorphic engineers to build sets for training spiking neural networks (SNN).

Keywords:
Neuromorphic engineering; spiking neural networks; Address-Event-Representation; neuro-inspired auditory sensor;

1. Introduction

Neuromorphic engineering is a discipline that studies, designs and implements hardware and software that tries to mimic the way that nervous systems work, focusing its main inspiration in how the brain solve easily complex problems. Currently, the neuromorphic community has a set of neuromorphic hardware, as sensors [1][2], learning circuits [3][4][5], neuromorphic information filters and features extractors [6][7], robotic and motor controllers [8][9][10][11]. In the field of neuromorphic sensors, diverse neuromorphic cochleae can be found [2][12][13]. These sensors are able to decompose the audio in frequency bands, and represent them as streams of short pulses, called spikes, using the Address-Event Representation [14] to interface with others neuromorphic layers. On the other hand, there are several software tools in the community, as NENGO [15] or BRIAN [16] for spiking neural networks simulation with or without learning; or jAER [17] for real-time visualization and software processing of AER streams captured from the hardware using specialized interfaces [18]. The software presented in this paper, called NAVIS, has the aim to help neuromorphic community to work with cochleae data in order to visualize and adapt this information to build training sets for later learning. To demonstrate these software functionalities a 64-channel binaural Neuromorphic Auditory Sensor (NAS) for FPGA [13] has been used together with an USB-AER interface [18], as it can be seen on Fig. 1. NAS response is stored as aedat files through jAER. Nevertheless, since this software works with aedat files, it can work with any cochleae sensor connected to jAER.

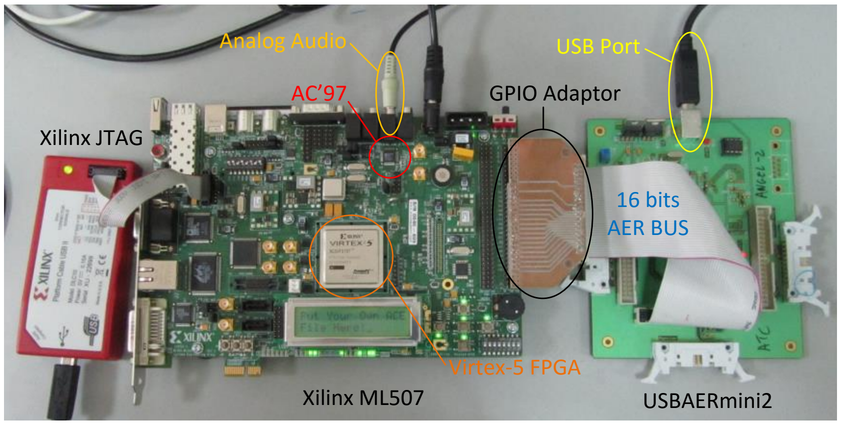

Fig. 1: 64-channel binaural NAS connected to an USB-AER mini2 board

NAVIS gives to neuromorphic engineers a set of functions: (1) detailed data visualization through cochleograms, sonograms, histograms, average activity and channels disparity; (2) it performs a set of algorithms for different streams splitting; (3) it can removes silences and generates a set of new files with the most relevant auditory information; (4) it can be a test bed for building data sets for spiking neural networks training. For example, it can reproduce an audio file, with all the cases and samples to train a Spiking Neural Network (SNN), i.e. SpiNNaker [19], and it can store this information into aedat files to be used in jAER. After recording an aedat file, NAVIS is used to visualize features and measure the quality of the auditory information, it can search silences between samples and use them to split the original file into several streams with short sentences or words; and, finally, it can store a new set of aedat files without silences and providing one file per sample.

The rest of the paper is structured as follows: section II presents software functionalities with results obtained under our testing scenario. Then, section III presents the performance analysis, running the software on three different computers to study its scalability. Finally, section IV presents the conclusions, summarizing what have been achieved with this work.

2 Software Framework

2.1 Software Functionalities

Neuromorphic Auditory VISualizer Tool (NAVIS Tool) is able to read aedat files in order to perform a set of algorithms to the binary contained data to obtain results and graphical representations that will provide relevant information about the events communicated during a time period. Once the file is loaded, it gives a set of functions through the menus and the left side tool bar, which can be applied to the data. The stream of events contained in the aedat file used in the presented examples corresponds to a young woman reading the first sentence from the famous Spanish novel *The Ingenious Gentleman Don Quixote of La Mancha*: "*En un lugar de La Mancha*". The main window of the application can be seen in Fig. 2, which shows the cochleogram for the loaded aedat file and a sidebar with currently available developed tools. Following subsections detail each of these functions and its result for this *Quixote* scenario.

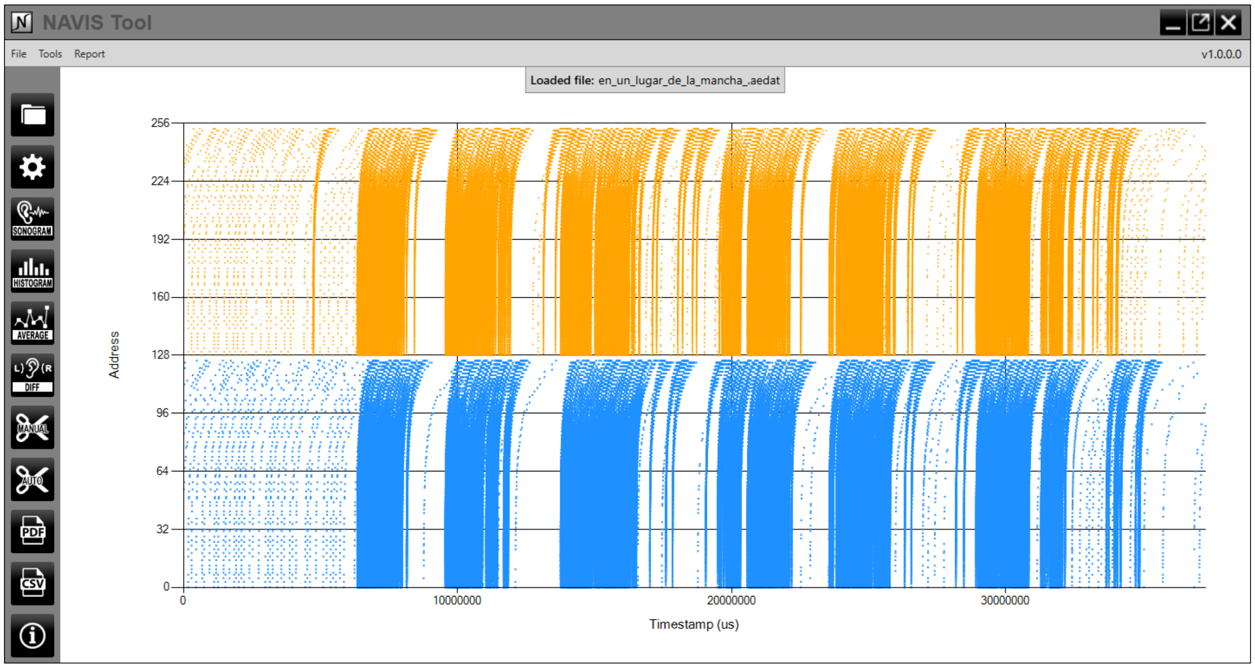
Loaded file: en_un_lugar_de_la_mancha_.aedat

Fig. 2: Cochleogram representing "En un lugar de La Mancha", obtained with NAVIS Tool. Left 64-channels output events are represented in blue and right ones in orange. Colors can be modified by the user.

2.1.1. Cochleogram

After choosing the aedat file to be loaded, the internal data decodification process and the subsequent cochleogram calculation and representation in the main application window are launched. The output can be seen in Fig. 2, where X axis represents time (µs) and, the Y axis is the AER address assigned to each frequency band. Each dot corresponds to an event that has been fired in a particular AER address at a specific time. The picture shows the AER events fired for both cochleae: the left one at the bottom and the right one on top, embracing up to 256 addresses for the two sets of 64 channels of the binaural NAS [12], where each channel has positive and negative events. The address event structure is shown in Fig. 3 for a 16-bit AER bus.

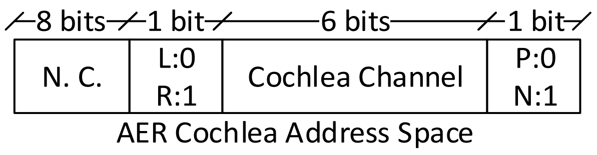

Fig. 3: Address structure for the AER events. 8 most significant bits are not used, depending on the address size of the aedat file. Only 8 LSB are used: most significant bit to select left ('0') or right ('1') cochlea, 6 to determine the cochlea channel and the less significant bit to choose between the positive ('0') or negative ('1') event.

Lower addresses belong to the events of higher frequency channels. The response of the events appears to be “lying”, this is because of the cascade architecture of the internal filters (Spike Low Pass Filter –SLPF– [6]), since each filter induces a small phase increment, which increases as the spikes go across a SLPF [13].

2.1.2. Sonogram

Fig. 4 represents the spike rate of both left and right cochlea in a color map, where X axis is time, Y axis is the cochlea channel, and the color is the spike rate of the channel in a particular time period, software settable. Each one of the words contained in the sentence “*En un lugar de La Mancha*” can be distinguished easily. The way colors are generated is: first the highest ratio of the sonogram is calculated and, after that, every single one is divided by that value, so that we obtain numbers between 0 and 1. These numbers can be divided in two subsets: the first one corresponds to numbers greater or equal than 0.5, which are scaled in a red-green gradient; and the second one contains numbers below 0.5, which are scaled in a green-blue gradient. Red represents the highest spike rate and blue the lowest one.

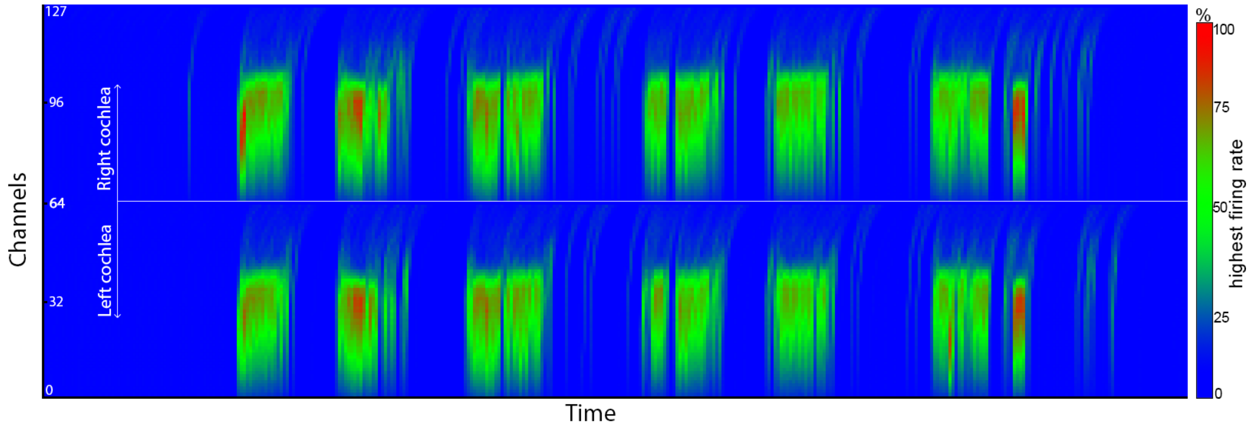

Fig. 4: Sonogram

2.1.3. Histogram

For certain applications it is important to know which cochlea channels are the ones that fire more events. The histogram is the appropriate chart to measure this information. Fig. 5 shows the result of the histogram for the aedat file that is being used in the test scenario. The X axis represents the 256 possible AER address values, while the Y axis is the number of events fired. An option is included for the user to choose the histogram to be normalized or not, as shown in Fig. 5.

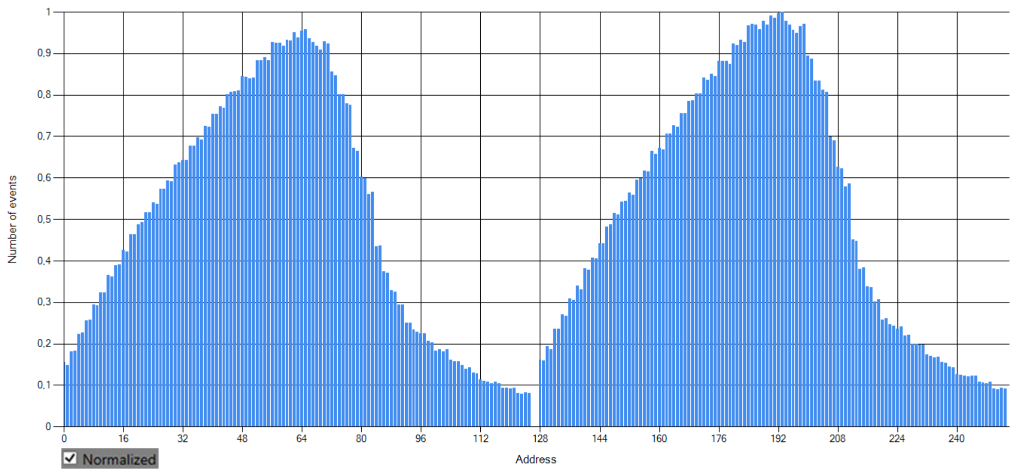

Fig. 5: Histogram

2.1.4. Disparity between left and right channels

The tool allows the user to know the difference between the left and the right cochleae, so that the predominant one in a particular time period can be identified. This will be extremely useful for source echolocation because dominant events polarity and spike rate after this operation can directly determine if the sound source is on the left or right of the sensor and how far it is after a certain calibration process or a learning algorithm.

The way this output is generated is similar to the sonogram process but subtracting the values corresponding to the same channel in the left and right cochlea to know the difference between them in an absolute value. Red tones in the graph means that the left cochlea is being predominant in that time period, and so with the green tones and the right cochlea. The output of this function for the aedat file that we have been using in this paper can be seen in Fig. 6.

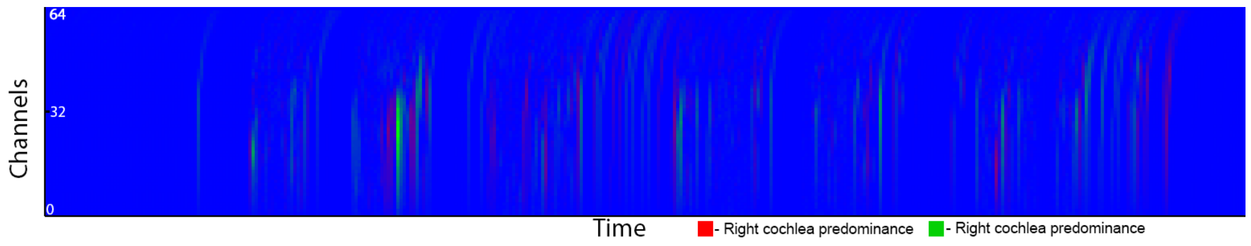

Fig. 6: Disparity between the left and the right cochleae

2.1.5. Average activity of both channels

It is really important to know the activity of the cochleae in terms of number of AER events fired per second in a particular time period, either for source echolocation as the tool presented above or to quantify the firing rate of both cochleae. This tool provides this functionality, generating a 2-axis chart, Fig. 7, where the X axis represents time (timestamp, µs) and the Y axis is the number of mega-events (10^6 events) fired per second, calculated by counting the number of events produced between a time period (integration period) and dividing it by this number.

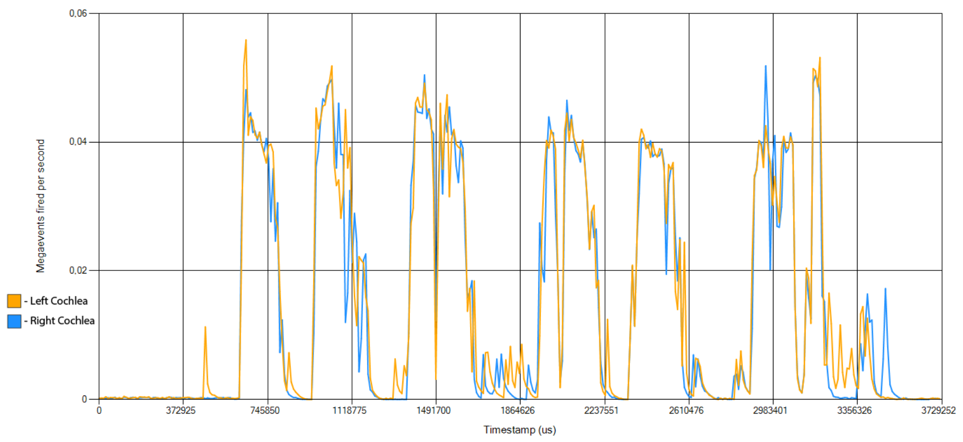

Fig. 7: Average activity of both cochleae

2.1.6. Aedat files split

One of the main aims of this application was to be able to split it an aedat file automatically into other different files by analyzing the fire rate of both left and right channels. This analysis in done over a period of time and the process stablishes a split or division when there is not enough activity to be considered as a sound while removing silences. These sections can be saved as single aedat files that could be loaded and used by other neuromorphic systems. The main application for this utility is to build training sets for spiking neural networks (SNN), so that after recording into only one file a sequence of audio samples, it is able to distinguish all individual stimulus and store them in different files.

NAVIS allows the user to manually extract a specific section of the original aedat file and save it as a separate file. In addition, it implements a function that allows splitting the file automatically based on the averaged activity level of the channels. Two configurable parameters are needed for this:

- The Noise Threshold: it is the percentage value from the total number of events fired in the aedat file that an integration period needs to have to be considered as sound instead of noise from the transition between two words.
- The Noise Tolerance: it tells for how many consecutive integration periods it is needed to detect a higher number of events than the percentage from the Noise Threshold to be considered as a real sound instead of noise.

Then, when a time period with not enough firing rate is detected, if the process has previously passed through Noise Tolerance consecutive times or more detecting sound, a new split is created, otherwise this information will be taken as noise and will be omitted in the results of this function.

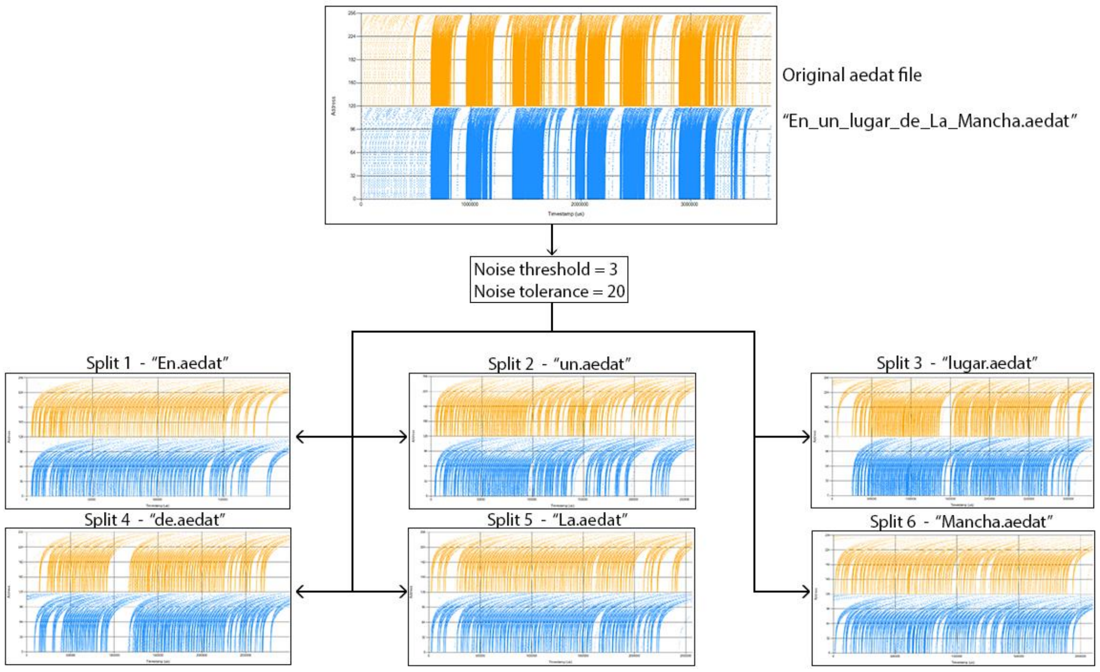

Fig. 8: Automatic aedat splitter output

Fig. 8 shows the output obtained after applying this functionality, using a value of 3 units for Noise Threshold and 20 units for Noise Tolerance. The process generated six new aedat files which correspond to each word of the sentence "*En un lugar de La Mancha*".

2.2. Results overview

Among the implemented functionalities, NAVIS Tool lets users save a final report of all the possible treatments that can be performed to the input file. This includes the cochleogram, histogram, sonogram, disparity between both cochleae and average activity. These can be either selected or not by the user, allowing him to obtain only the information of interest. The process takes a longer time to execute depending on the aedat length and how many functions are selected. In the next section we have measured the performance of the tool.

3. Performance Analysis

The information of each AER event in an aedat file is stored in an object that has three attributes: address, timestamp and an ID to know its position within the file. When a file is selected, the loading process adds each one of these objects to a list, which will contain the entire information of the events fired in the aedat file when this process ends (more information about the software architecture and data model can be found in the NAVIS wiki [20]). As large data size files (more than a million of events in this example) are usually used in this application, software optimization techniques need to be used in order to work with the information in a fast way and reduce the CPU's load as much as possible. Therefore, LINQ (Language-Integrated Query) [21] query expressions have been used. LINQ is a component from the Microsoft .NET platform [22] that offers an API called Standard Query Operator (SQO) that has a set of operators that have let us handle large lists, like lists of events (aedat files), in the most optimized way. Using LINQ functions, which allows to handle complex algorithms on list structures without using nested loops, has made most of the functionalities of NAVIS able to execute in a low execution time, which would not be possible without these operators.

3.1. Study on execution times

TABLE I shows execution times for some functions that have been implemented in NAVIS Tool, measured with Microsoft Visual Studio Profiling Tools [23]. For this study three different processors have been used, so that the scalability of the operations can be studied. In addition to the total time calculated from adding the execution times for each function, the table also presents the percentage from the total time and the time per event for every tool. This last number is extremely important due to the fact that the execution time directly depends on the total number of events that the file has. We can observe that generating the pdf report is the function that takes the most time to execute (around 45% of the total), since the process to create that file needs to calculate the output from the rest of the functions.

TABLE I: Comparison on execution times per function between three different processors

Functions	**Processors**					
	Celeron P4500 (1.87GHz)		***i5-4460 (3.2GHz)***		***i7-4770 (3.5GHz)***	
	Total time percentage	***Time per event (ms)***	***Total time percentage***	***Time per event (ms)***	***Total time percentage***	***Time per event (ms)***
Load Aedat	5.293%	$30.96*10^{-4}$	5.556%	$11.72*10^{-4}$	6.346%	$12.70*10^{-4}$
Sonogram	14.05%	$82.21*10^{-4}$	14.60%	$30.80*10^{-4}$	13.859%	$27.76*10^{-4}$
Histogram	0.112%	$0.657*10^{-4}$	0.127%	$0.268*10^{-4}$	0.166%	$0.33*10^{-4}$
Disparity between cochleae	13.71%	$80.11*10^{-4}$	13.67%	$28.83*10^{-4}$	13.676%	$27.39*10^{-4}$
Average activity	6.484%	$37.92*10^{-4}$	5.648%	$11.91*10^{-4}$	5.942%	$11.90*10^{-4}$
Automatic split	15.88%	$92.87*10^{-4}$	15.60%	$32.91*10^{-4}$	15.19%	$30.43*10^{-4}$
Manual split	0.196%	$0.114*10^{-4}$	0.237%	$0.499*10^{-4}$	0.528%	$1.05*10^{-4}$
Generate PDF	44.28%	$258.9*10^{-4}$	44.56%	$93.98*10^{-4}$	44.293%	$88.74*10^{-4}$
Total time	100% (63230 ms)	$584*10^{-4}$ ms	100% (22804 ms)	$211*10^{-4}$ ms	100% (21660 ms)	$200.3*10^{-4}$ ms

4. Conclusion

In this paper we have presented the design and development details of a software application that is able to load AER streams stored in aedat files obtained from a 64-channel binaural NAS, captured using jAER software and USB-AER hardware interface. A data model has been designed to store and use the information extracted from aedat files. This model represents the information efficiently and facilitates its subsequent access and processing.

A set of functionalities have been designed to apply operations to the data gathered from the aedat file and to generate graph results. To improve execution times and reduce resources used in memory, these functions have been optimized using LINQ query expressions and lambda functions.

Finally, the tool includes a functionality able to split an aedat file into several ones, both manually and automatically, depending on the average activity of the cochlea. This is a very useful capability for creating training sets for spiking neural networks.

The presented software tool helps neuromorphic researchers to process and evaluate large amounts of cochlear information easily and efficiently.

***Acknowledgements**

We want to thank Alejandro Linares-Barranco for his support and contribution on this work, which is supported by a Spanish government grant (with support from the European Regional Development Fund) for project BIOSENSE (TEC2012-37868-C04-02).

References

[1] Lichtsteiner, P.; Posch, C.; Delbruck, T. A 128×128 120dB 15 µs Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal on Solid-State Circuits*. 2008, *43*, 566-576.

[2] Chan, V.; Liu, S.C.; van Schaik, A. AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface. *IEEE Trans. Circuits and Systems*. 2007, *54*, 48-59.

[3] Serrano-Gotarredona, R.; Serrano-Gotarredona, T.; Acosta-Jimenez, A.; Serrano-Gotarredona, C.; Perez-Carrasco, J.A.; Linares-Barranco, B.; Linares-Barranco, A.; Jimenez-Moreno, G.; Civit-Ballcels, A. On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing. *IEEE Trans. Neural Network*. 2008, *19*, 1196-1219.

[4] Hafliger, P. Adaptive WTA with an Analog VLSI Neuromorphic Learning Chip. *IEEE Trans. Neural Networks.* 2007, *18*, 551-572.

[5] Indiveri, G.; Chicca, E.; Douglas, R. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks* 2006, *17*, 211-221.

[6] Jimenez-Fernandez, A.; Linares-Barranco, A.; Paz-Vicente, R.; Jiménez, G.; Civit, A. Building blocks for spikes signals processing. In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), 18–23 July 2010; pp. 1-8.

[7] Linares-Barranco, A; Gomez-Rodriguez, F; Villanueva, V; Longinotti, L; Delbruck, T. "A USB3.0 FPGA event-based filtering and tracking framework for dynamic vision sensors," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2417–2420.

[8] Linares-Barranco, A.; Gomez-Rodriguez, F.; Jimenez-Fernandez, A.; Delbruck, T.; Lichtensteiner, P. Using FPGA for visuo-motor control with a silicon retina and a humanoid robot. In *Proceedings of IEEE International Symposium on Circuits and Systems*, ISCAS, New Orleans, LA, USA, 27-30 May 2007; pp. 1192- 1195.

[9] Jimenez-Fernandez, A.; Linares-Barranco, A.; Paz-Vicente, R.; Jimenez-Moreno, G.; Berner, R. Spike-based control monitoring and analysis with Address Event Representation. In *Proceedings of IEEE/ACS Int. Conference on Computer Systems and Applications*, AICCSA, Rabat, Morocco, 10-13 May 2009; pp. 900- 906.

[10] Jimenez-Fernandez, A.; Jimenez-Moreno, G.; Linares-Barranco, A.; Dominguez-Morales, M.J.; Paz-Vicente, R.; Civit-Balcells, A. A neuro-inspired spike-based PID motor controller for multi-motor robots with low cost FPGAs. Sensors. 2012, *12,* 3831-3856.

[11] Perez-Peña, F.; Morgado-Estevez, A.; Linares-Barranco, A.; Jimenez-Fernandez, A.; Gomez-Rodriguez, F.; Jimenez-Moreno, G.; Lopez-Coronado, J. Neuro-Inspired Spike-Based Motion: From Dynamic Vision Sensor to Robot Motor Open-Loop Control through Spike-VITE. *Sensors* 2013, *13*, 15805-15832.

[12] Hamilton, T.J.; Jin, C.; van Schaik, A.; Tapson, J. An Active 2-D Silicon Cochlea. *IEEE Trans. Biomedical Circuits and Systems.*2008, *2*, 30-43.

[13] Jimenez-Fernandez, A; Cerezuela-Escudero, E; Miro-Amarante, L; Dominguez-Morales, M; Gomez-Rodriguez, F; Linares-Barranco, A; Jimenez-Moreno, G. A Binaural Neuromorphic Auditory Sensor for FPGA: A Spike Signal Processing Approach. Submitted to IEEE Transaction on Neural Networks and Learning Systems. Under review.

[14] Boahen, K. Point-to-Point Connectivity Between 0Neuromorphic Chips Using Address Events. *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*. 2000, *47*, 416-434.

[15] Bekolay, T; Bergstra, J; Hunsberger, E; DeWolf, T; Stewart, T.C.; Rasmussen, D; Choo, X; Voelker, A.R. Eliasmith, C. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics,* 2014, *7*, 48.

[16] Goodman, D; Brette, R. Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics* 2008, *2*, 5.

[17] jAER Open-Source Software Project. Available online: http://sourceforge.net/p/jaer/wiki.

[18] Berner, R.; Delbruck, T.; Civit-Balcells, A.; Linares-Barranco, A. A 5 Meps $100 USB2.0 Address-Event Monitor-Sequencer Interface. In *Proceedings of IEEE International Symposium on Circuits and Systems,* ISCAS, New Orleans, LA, USA, 27-30 May 2007; pp. 2451-2454.

[19] Painkras, E.; Plana, L.A.; Garside, J.; Temple, S.; Davidson, S.; Pepper, J.; Clark, D.; Patterson, C.; Furber, S. SpiNNaker: A multi-core System-on-Chip for massively-parallel neural net simulation. *IEEE Custom Integrated Circuits Conference (CICC),* 2012.

[20] NAVIS Tool Wiki. Available online: https://github.com/jpdominguez/NAVIS-Tool/wiki/Software-architecture.

[21] LINQ: .NET Language-Integrated Query. Available online: https://msdn.microsoft.com/en-us/library/bb308959.aspx.

[22] Microsoft .NET Framework. Available online: https://www.microsoft.com/net.

[23] Microsoft Visual Studio. Available online: https://www.visualstudio.com.

B- Required Metadata

B1 Current executable software version

Table 1 – Software metadata

Nr	(executable) Software metadata description	*Please fill in this column*
S1	Current software version	*1.0.0.0.*
S2	Permanent link to executables of this version	*https://github.com/jpdominguez/NAVIS-Tool/tree/master/NAVIS/NAVIS_LatestBuild*
S3	Legal Software License	*GNU General Public License (GPL)*
S4	Computing platform / Operating System	*Microsoft Windows*
S5	Installation requirements & dependencies	*Microsoft .NET Framework 4.5 or greater*
S6	If available Link to user manual - if formally published include a reference to the publication in the reference list	*https://github.com/jpdominguez/NAVIS-Tool/wiki*
S6	Support email for questions	*jpdominguez@atc.us.es*

B2 Current code version

Table 2 – Code metadata

Nr	Code metadata description	*Please fill in this column*
C1	Current Code version	*1.0.0.0.*
C2	Permanent link to code / repository used of this code version	*https://github.com/jpdominguez/NAVIS-Tool/tree/master/NAVIS/NAVIS_VisualStudioProject*
C3	Legal Code License	*GNU General Public License (GPL)*
C4	Code Versioning system used	*Git*
C5	Software Code Language used	*C Sharp (C#)*
C6	Compilation requirements, Operating environments & dependencies	*Microsoft Visual Studio, Microsoft .NET Framework 4.5 or greater*
C7	If available Link to developer documentation / manual	*https://github.com/jpdominguez/NAVIS-Tool/blob/master/NAVIS/NAVIS_LatestBuild/NAVIS.XML , https://github.com/jpdominguez/NAVIS-Tool/wiki/Software-architecture*
C8	Support email for questions	*jpdominguez@atc.us.es*