FPGA Implementation Of A Seizure Detector Using Spectral Domain Feature Extraction

Marin Bricq 315273
Robotic master student

Professor:
Alexandre Schmid

5th June 2023

Abstract

This paper describes a patient-specific algorithm for classification of epileptic seizures using intracranial electroencephalography (iEEG). The method consists of extracting statistical features from the average signal of all electrodes and its discrete wavelet transform (DWT) coefficients. A total of forty features is computed and used as the input of a linear support vector machine (SVM) classifier. The data for testing consist of 25 seizures from 16 different patients. It is shown the algorithm reaches a sensitivity of 98.0%, a specificity of 95.5% and a delay of 17.1 seconds. Finally, it is demonstrated that the algorithm is compatible with FPGA implementation by testing on 3 randomly selected patients.

1 Introduction

1.1 Epilepsy

Epilepsy is a chronic neurological disorder that affects around 1% of the world's population. About 50 million people suffer from it [1]. It is defined as an abnormal electrical activity of the neurons in the brain which might have catastrophic consequences. Around a third of the patients' seizures are said to be drug-resistant, which means that they cannot be stopped using a classical treatment [2]. For these patients, it is capital to monitor them to avoid lethal consequences. A device that would be able to detect or even predict them would remove a weight on the patient. That device needs to be small and computationally light as it would be carried at all time.

1.2 Seizure detection

The main idea behind detection is to look at the electrical activity of the brain through an electroencephalogram (EEG). The task is to find features and a classifier that will separate the samples in two categories: either ictal (seizure) or non-ictal phase. In the literature, various features have been used for detection such as statistics computed from the Fast Fourier Transform [3] and [4], but EEG signals are non-stationary and thus it is not the most effective. For this reason, other researches have focused on using the Wavelet transform instead [5], [6], [7] and [8]. After computing the wavelet coefficients, one can compute statistics such as mean, energy, standard deviation and others. These values will for certain patients highly change between ictal or non-ictal epochs and thus allow to detect a seizure. Finally, the same statistics can be computed from the original signal directly without applying a transform [9], [10] and [11]. In this research, it was decided to use features from Wavelet transform and the original signal as it was shown that a detector using them can get really good results, as high as 99% accuracy for Xie et Krishnan [8].

After the features have been extracted, they will be given to a classifier which roles is to decide if the patient is currently suffering from a seizure or not. In the literature, three main classifiers were used along with wavelets features. First there are Random forest proposed by Mursalin et al. [12] which give good results. Übeyli [13] showed that neural network can also be used as a classifier and get up to 94% of accuracy,. Finally, Faust et al. [6] did a review of various papers using wavelet transform for epilepsy detection and many used a support vector machine (SVM) which showed good results. An advantage of SVM over the two others is the fact that it requires less computation and thus will consume less power. For this paper, the final system would be implemented on a FPGA board and for this reason an algorithm light in computation would be better. Thus, the choice was to use a linear SVM.

A diagram of the entire system built can be seen in figure 1, the rest of the paper will explain each of its elements, the results it got and will finally discuss them.

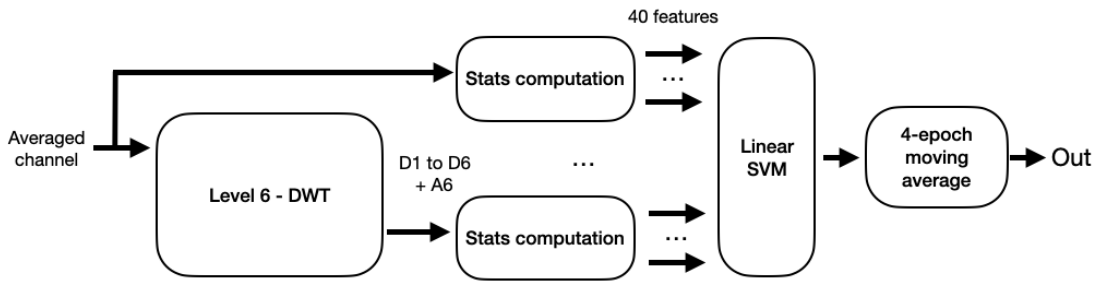

Figure 1: Diagram of the entire system, the features extraction is done by the DWT and computation of statistics, the linear SVM is the classifier.

2 Detection process

2.1 Dataset

The short-term dataset used in this paper is from the Sleep-Wake-Epilepsy-Center (SWEC) of the University Department of Neurology at the Inselspital Bern and the Integrated Systems Laboratory of the ETH Zürich, also

used in their research [14]. It is composed of various seizures from 16 different patients. The data was first pre-filtered between 0.5 and 150 Hz and then digitally converted on 16 bits at a rate of 512 Hz. All recordings have been verified by a certified EEG board-certified experienced epileptologist (K.S.) for identification of seizure onsets and endings. The number of seizures of each patient vary from 2 up to 13. For each seizure, the data is composed of a 3-minute pre-ictal segment, a 3-minute post-ictal one and the seizure (lasting between 10 and 1002 seconds). Originally the data contain each channel of recordings from all the electrodes, for this paper it was decided to keep only the average of all the channels for each seizure.

2.2 Discrete wavelet transform

The discrete wavelet transform (DWT) is a mathematical tool that allows to do a time-frequency analysis of a given signal. One of its main advantage over the Fourier transform is the ability to study non-stationary signals such as EEG. It decomposes the input signal into various sub-bands, each at a different scale thus capturing different information. The computation of the level J DWT is done mathematically as follows:

$$Y(t) = \sum_{i=0}^{J-1} \sum_{j=0}^{2^i-1} d_{ij} \psi_{ij}(t) \tag{1}$$

Where d_{ij} are the DWT coefficients and the basis function ψ_{ij} is built from a mother wavelet ψ as such $\psi_{ij}(t) = 2^{-i/2}\psi(2^{-i}t - j)$. The subscript i is the dilation and j is the translation of the wavelet.

The implementation that was chosen to compute these coefficients is the filterbank one as it is easy to do and fast. The input signal is filtered by two separate finite impulse response filters: the scaling filter which acts as a lowpass filter and the wavelet filter which acts as a high pass, both are also downscaled as they represent complementary frequencies keeping all the coefficients would be redundant. This gives two new signals: the approximation (low frequencies) and the details (high frequencies). They are said to be of level one. Then the same process can be repeated using the approximation as the new input, giving the approximation and details of level 2 (a complete system can be seen in figure 2). Each new signal will be a reconstruction of the original one using a different frequency band.

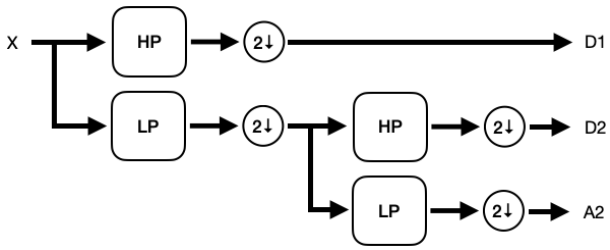

Figure 2: Block diagram for a level 2 DWT. X is the input signal, HP is a highpass filter, LP a lowpass filter. The outputs are details of level 1 (D1) and 2 (D2) and approximation of level 2 (A2).

This transform is broadly used in signal processing for different domain and a lot of different filters can be found in the literature. The review of Faust et al. [6] shows that the wavelet Daubechies 4 (db4) is often used for epilepsy detectors and was also used for this paper. At first, the DWT was computed up to level 8 but it appeared that when the machine learning was performed to find the optimal SVM no features of higher level than 6 were used, therefore the final system only went up to level 6. This implies that all features are computed for the original signal and 7 new sub-bands (6 details D1 to D6 and approximation A6). The results of the DWT implementation performed for this paper can be seen in figure 3.

Finally, the implementation itself was done on Simulink using the add-ons HDL Coder which will generate the VHDL code required for the FPGA implementation. Meyer-Baese et al. [15] showed how it could be achieved in details.

2.3 Statistics extraction

The features from the 8 different signals were computed based on epochs of $N = 512$ samples which correspond to a one-second epoch. It was chosen because 512 samples were enough to represent the information but one second is short enough to reduce the delay before detection. Dong et al. [16] and Guo et al. [17] showed that the following statistics were interesting for epilepsy detection: energy (E), standard deviation (σ), curve length (L), maximum (M) and minimum (m) of the signal. The mean (μ) of the epoch is also computed though not used for the features as it showed no advantages but required for standard deviation. The statistics are computed for each of the eight signals as seen in equations (2) to (6).

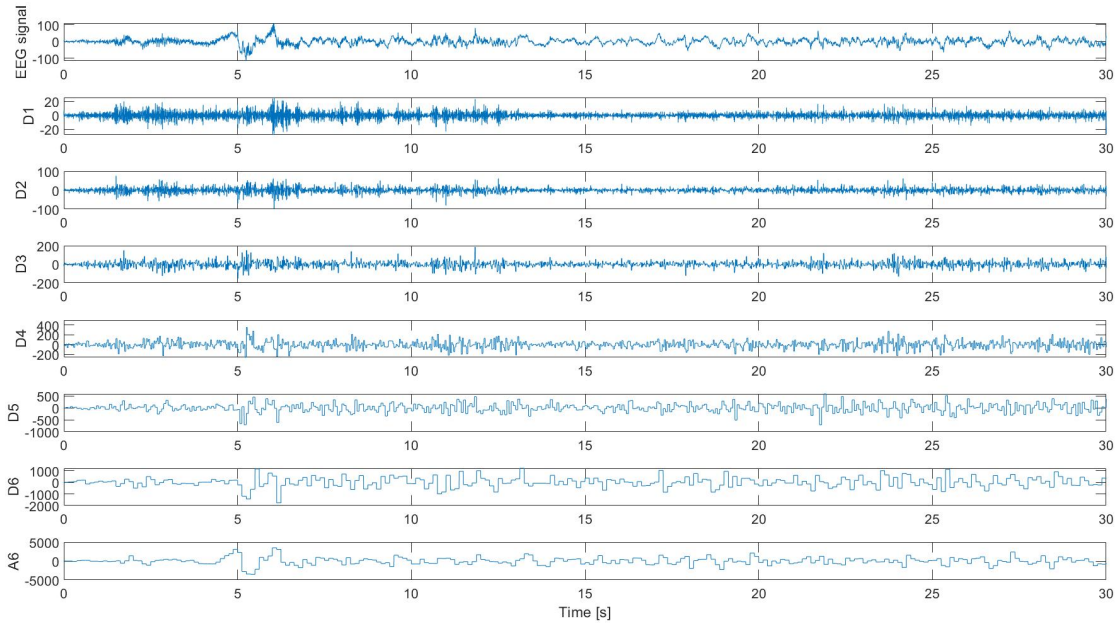

Figure 3: The different coefficients of the DWT were computed for a part of seizure 1 of patient 1. At the top, one can see the original EEG signal, it is 30 seconds taken right before the beginning of a seizure. The different details and the approximation can be visualized bellow. It allows to see that indeed the approximation gives the lowest frequencies (i.e. the envelope of the signal) while the details go into higher and higher frequencies (i.e. highly varying parts of the signal).

$$E = \frac{1}{N}\sum_{n=1}^{N} x_n^2 \tag{2}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_n - \mu_n)\cdot(x_n - \mu_{n-1})} \tag{3}$$

$$L = \sum_{n=1}^{N-1} |x_{n+1} - x_n| \tag{4}$$

$$M = \max_n x_n \tag{5}$$

$$m = \min_n x_n \tag{6}$$

The standard deviation and mean were computed using the Welford's online algorithm as revisited by Efanov et al. [18].

This gives a total of forty features to train the SVM. For each patient, the features were ranked by computing the average of each feature for the ictal epochs and dividing this by the average for the non-ictal ones. This was done on the training set (see section 2.4).

2.4 Linear SVM

The goal of linear SVM is to find an hyperplane that will separate a features space into two regions, it will then check for any datapoint in which sides it stands. To do so, it uses the training data to optimizes two parameters: β, the normal to the hyperplane, and b, the intercept also called bias. It also has a miss-classification cost matrix that indicates to which degree false positives or false negatives should be optimized. For the training, the computer will optimize the function seen in equation (7).

$$
\begin{aligned}
&J = \frac{1}{2}\|\beta\|^2 + C^+ \sum_{n\in C_1} \xi_n \; + \; C^- \sum_{m\in C_2} \xi_m \\
&\text{subject to } y_i(\beta^T x_i + b) \geq 1 - \xi_i, \; i = 1,...,N \\
&\xi_i \geq 0, \; i = 1,...,N
\end{aligned}
\tag{7}
$$

Where x_i is the training feature vectors, N is their number, y_i their associate class (1 or -1), ξ_i the slack variables and finally C^+ and C^- are the miss-classification costs associated with the positive (C_1) and negative class (C_2) respectively.

After the training, the model can be used to assess if any epoch is ictal or not using its features vector x:

$$
out = sign(\beta^T x + b) = \begin{cases} 1, & \text{if epoch is ictal} \\ -1, & \text{otherwise} \end{cases}
\tag{8}
$$

To obtain results which are not influenced by the differences in order of magnitude of the different features, they were first normalized. This is done as follows:

$$
\tilde{x}^j = \frac{x^j - \mu_x^j}{\sigma_x^j \cdot scale}
\tag{9}
$$

Where $\tilde{x}^j$ is one component of the normalized feature vector which is actually used in (7) and (8) instead of x. The values of μ_x^j (mean of the jth feature), σ_x^j (standard deviation of the jth feature) and $scale$ (intrinsic parameter of linear SVM) were computed on the training set.

For each patient, around 75% of the data were used for training purposes but always keeping at least one seizure for the testing. This gave enough data to train while still being able to verify no overfitting occurred. A greedy algorithm was implemented to choose which features to used based on previously mentioned ranking (in section 2.3). The following measures were used to assess the effectiveness of a model: sensitivity (number of seizures detected), specificity (number of true negatives over number of non-ictal epochs) and accuracy (average of both). At first, the algorithm added new features to the model and retrained it to maximize accuracy (the features kept for each patient is in appendix A). Afterwards, the models were manually tuned, using the miss-classification cost matrix, to minimize the number of false positives while still detect seizures and with minimum delay (the costs are in appendix B). Finally, it was noted that the predictor ($\beta^T x + b$) was varying a lot for certain patients which resulted in a lot of false outputs. A way to mitigate this is not to use directly this value in the *sign* function for equation (8) but instead to compute the moving average of the predictor over the last four epochs.

2.5 Implementation

For this paper, the explained work was implemented and tested in three different steps. First, to be able to implement it fast and to be able to build the models for each SVM it was done using MATLAB, these models were then saved and their parameters were reused in the two other implementations. Primary results were computed with this version in order to make sure the detector would work using a computer for all calculations. Then, the actual detector was programmed using Simulink and the HDL Coder library which allows to easily build a system that then can be implemented on a FPGA board. The implementation of the level 1 DWT and to compute the energy of a signal using these blocks can be seen in figure 4. The VHDL code is then automatically generated, the one corresponding to the level 1 DWT block is given in appendix C.

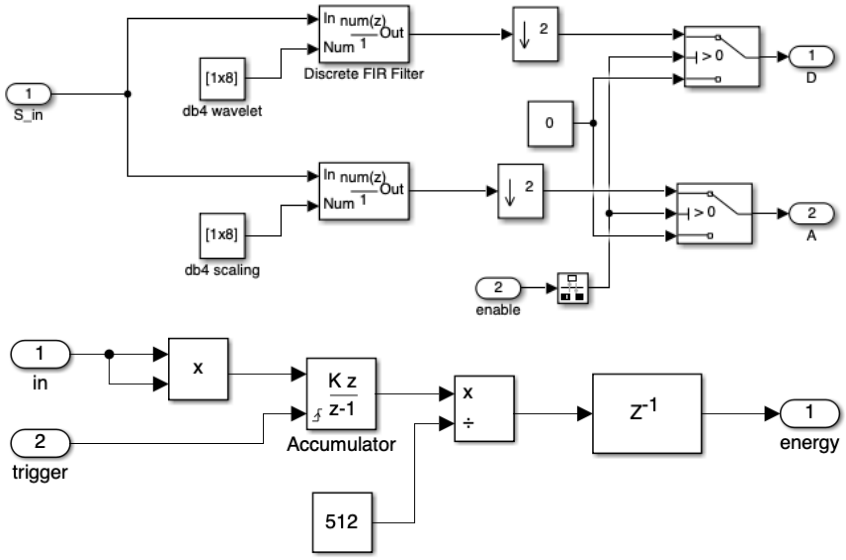

Figure 4: Top circuit: the two filter blocks that performs the DWT along with downsampling and a control block to enable the output. The values of the filters are stored in two memory blocks.
Bottom circuit: computation of the energy over 512 samples by accumulating in^2 and then dividing by $N = 512$. There is an added extra delay necessary to synchronize the computation of all statistics. The trigger is used as a reset of the accumulator every 512 samples.

An entire final predictor was built using a combination of these blocks and this allowed to re-test all seizures in Simulink to check it was working correctly. The conversion to HDL code was verified by building test benches. This can also be achieved directly in Simulink, it will automatically build testing files including the inputs and expected outputs. These can then be simulated on Modelsim for example and it will automatically check if the output is indeed the same as the expected one. These were done for five randomly selected patients and they all showed the exact same responses as the Simulink tests (an example can be seen in figure 5). This confirmed the system could be turned into HDL and tested on the final implementation.

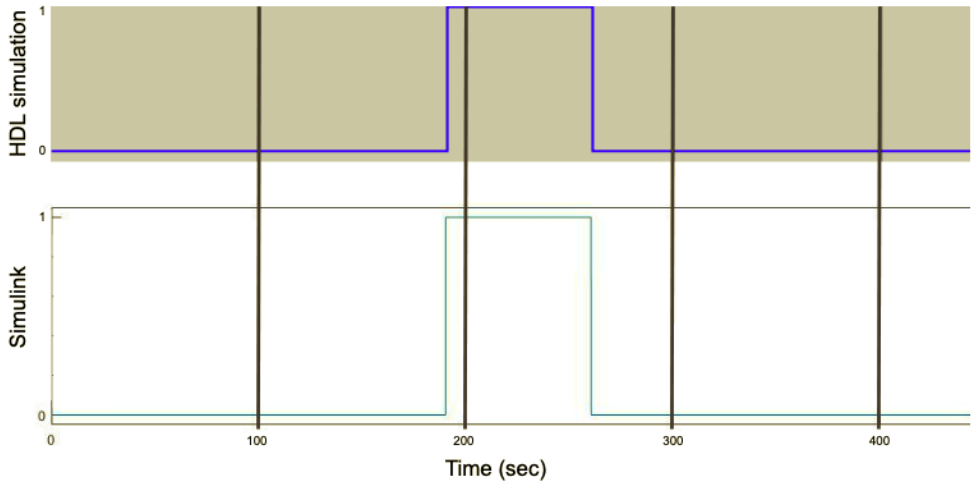

Figure 5: The output of the Simulink implementation and tested HDL code for the same seizure (which is seizure 2 of patient 3). This allows to visually confirms the automatic verification that indeed both outputs are equal.

Finally, a test for the system was built on the Intel D2-115 board using a Nios II CPU to communicate with the computer in order to check the results. This final test was realized to prove that the detector could indeed work and was performed for only three randomly selected patients. For this test, a simple system was built using the softwares QSys and Quartus 17.1 from Intel. This system consisted of the Nios II CPU, a JTAG UART module for debugging (built-in by Intel), a RS-232 UART module (included in the Intel University program) to communicate with MATLAB, an on-chip memory, some LEDs for visual feedback and the interface to the predictor (figure 6 shows a diagram of such system). This interface is required as the system uses an Avalon bus for communication. The basic working concept was the following: first, the Nios II CPU sends to the predictor the parameters of the SVM (vector β, b, *scale*, all μ_x and σ_x for each feature) and when it is done it turns on a LED. MATLAB can now send the data from the seizures using the RS-232 protocol. The CPU would catch and store them, once it received 512 samples it writes them to the interface's internal registers and then writes a start signal. The interface then

sends the samples to the actual predictor block which computes the output. When it is done the interface saves the output and keeps a register on to indicate so. The CPU regularly polls this register until it is equal to 1. Finally, it can read the output value and send it to the computer. MATLAB stores it and the cycle continues. The finite state machines controlling this system are given in appendix D and some part of the VHDL and C codes used to make it work are in appendix E.

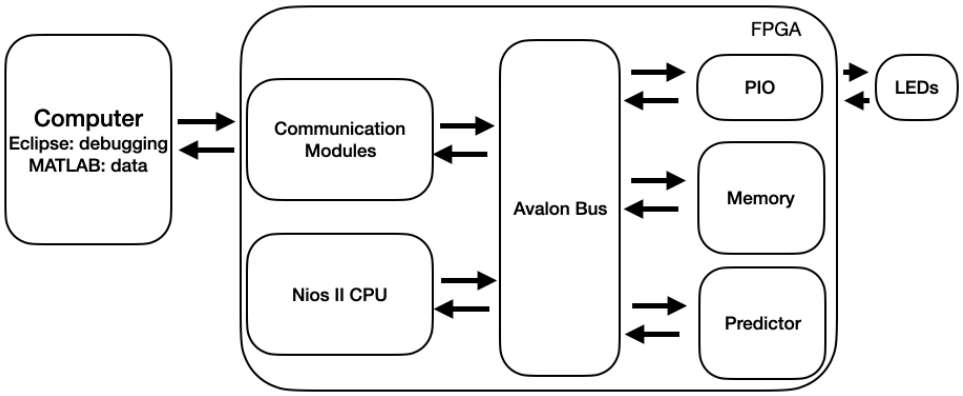

Figure 6: Diagram of the entire system built on the Intel D2-115 board.

3 Results and discussion

3.1 Results

The criteria that were used to determine if the results are good are the following:

1. **Seizure sensitivity:** number of seizures detected divided by total number of seizures.
2. **Specificity:** number of true negatives divided by total number of non-ictal epochs.
3. **Classification accuracy:** average of seizure sensitivity and specificity.
4. **Delay:** time between the moment a seizures is detected and the moment the epileptologist indicated the seizure started (the value was kept above 0 if the seizure was detected in advance).

These criteria were computed for all patients and for the three different testing phase of the algorithm as a manner of comparison. For MATLAB and Simulink, they are given in table 1.

Table 1: Performances for each patient on the two first test phases.

ID	Number of seizures	MATLAB				Simulink			
		Sensitivity (%)	Specificity (%)	Accuracy (%)	Delay (sec)	Sensitivity (%)	Specificity (%)	Accuracy (%)	Delay (sec)
1	3	67.7	96.1	81.9	2.5	67.7	94.6	81.2	4.3
2	1	100.0	97.5	98.8	0.0	100.0	97.8	98.9	1.0
3	1	100.0	99.7	99.9	8.0	100.0	99.4	99.7	6.0
4	2	100.0	70.1	85.1	12.5	100.0	74.3	87.2	16.4
5	3	100.0	99.1	99.6	34.7	100.0	98.1	99.1	33.7
6	1	100.0	96.9	98.5	0.0	100.0	94.4	97.2	0.0
7	1	100.0	98.9	99.5	7.0	100.0	97.8	98.9	11.0
8	1	100.0	97.2	98.6	7.0	100.0	97.5	98.8	10.0
9	2	100.0	97.1	98.6	35.0	100.0	98.3	99.2	38.0
10	1	100.0	100.0	100.0	3.0	100.0	99.4	99.7	4.8
11	1	100.0	100.0	100.0	21.0	100.0	99.4	99.7	24.0
12	3	100.0	97.9	99.0	4.7	100.0	98.0	99.0	15.3
13	1	100.0	98.1	99.1	5.0	100.0	91.7	95.9	6.5
14	1	100.0	98.3	99.2	9.0	100.0	94.7	97.4	9.0
15	1	100.0	98.9	99.5	74.0	100.0	97.2	98.6	75.0
16	2	100.0	96.1	98.1	15.0	100.0	95.4	97.7	18.0

They are computed only for the testing dataset, which is composed of approximately 25% of the entire dataset. The result using the first implementation showed a specificity above 95% for all patients (except for patient 4) with an average of 97.2%. Furthermore, only one seizure was not detected (seizure 13 of patient 1). Sensitivity has an average of 98% and accuracy of 97.2%. Finally, the delay is on average of 14.9 seconds, note that for a single patient the given delay is the average over all tested seizures.

For the second phase, the sensitivity got the same results as before, meaning an average of 98%. The specificity is a bit lower but still above 91.7% for all patients, again except patient 4. The average specificity is at 95.5%. The accuracy thus also went lower, with a new average of 96.7%. The delay of detection is a bit higher and up to 17.1 seconds.

Finally, for the actual test on the FPGA three patients were randomly selected to make sure the detector also worked. Those were patients 2, 3 and 7. For all of them, the testing dataset consisted of one seizure. The results are given in table 2. This shows an average of 100.0% of sensitivity. The specificity is at an average of 97.1% with a minimum of 91.9%. The accuracy is at 98.6%. Finally, the average delay is of 5.3 seconds. These results can be compared to the same patients for the MATLAB test. They had a sensitivity average of 100.0%, an average specificity 98.7% and accuracy of 99.4%. Finally, the delay was of 5.0 seconds.

Table 2: Performances of the FPGA detector on three patients.

ID	Sensitivity (%)	Specificity (%)	Accuracy (%)	Delay (sec)
2	100.0	91.9	96.0	3.0
3	100.0	100.0	100.0	4.0
7	100.0	99.4	99.7	9.0

The outputs of the three implementation for seizure 2 of patient 3 can be visualized in the figure 7. It shows why the specificty is not at 100.0% for the MATLAB and Simulink tests while being for the FPGA. One can also note that the seizure is not constantly detected by the FPGA but what is required is that it detects it once.

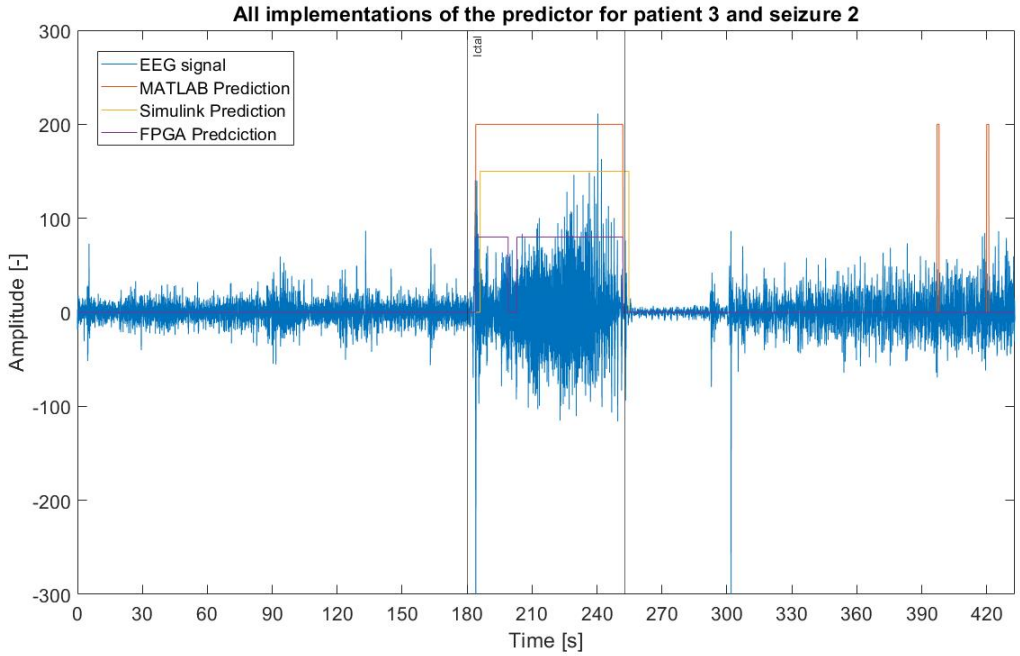

Figure 7: The output of the three implementation for seizure 2 of patient 3. The value of the predictors' output have been changed for the three to display them more clearly, the important information is when the level is high it detects a seizure. The two dark vertical lines are the indications of the seizure as given by the expert.

3.2 Discussion

This method showed almost perfect sensitivity in the three implementations with all but one seizures detected, this proves the combination of chosen features along with linear SVM has a true potential for epileptic seizure detection. In the first stage of testing the specificity was above 95% which shows it is capable to distinguish a seizure with a low rate of false positive. Furthermore, the delay was of 15 seconds on average which is good for this application as it was shown by Hirsch et al. [19] that seizures appears on iEEG 20 seconds before clinical symptoms. The main drawback is that the detector is sometimes unable to work correctly as indicated with the results for patient 4 with the specificity going down to 70.1%. This might be due to the fact that epilepsy can either show local or global patterns but in this paper the input was always an average of all channels of the EEG which lose local information. As in this dataset it is only the case for one patient and the average accuracy is still at 97.2%, it was considered that doing a more extensive computation of features was not necessary. Another point that can be noted, which is also true for the Simulink and FPGA results, is that there is a compromise between the delay and specificity. Indeed, by increasing the miss-classification cost associated with the non-ictal class (C^-) one can reduce the number of false positive and thus increase the specificity. The drawback is that this will increase the number of false negative epochs and thus the seizures will most probably be detected later, in extreme cases it could stop the detection entirely. This could be looked into further in a following research to obtain better results.

The implementation using Simulink has a lower specificity, the average being at 95.5% against 96.4%. This result is still considered good, it is explained by the difference in the computation, this new model was built with the goal of a FPGA target and for this it required to limit the number of bits used thus some errors. For the same reason, the accuracy went down to 96.7%. The delay also went a bit higher to 17.1 seconds but it still lower than the goal of 20 seconds.

Finally, for the actual FPGA tests the results are very close to the ones from the original test phase for these three patients. This shows that the system designed for this paper can indeed be implemented on a FPGA board for low consumption and is able to detect most seizures in less than 20 seconds with a low false positive rate.

The results were compared to the paper of Burello et al. [20] who also worked on the SWEC-ETHZ database. They worked by combining local binary patterns with hyperdimensional computing. It was done using Python code and thus can be compared with the MATLAB results from this paper. They obtained a sensitivity of 99.4%, a specificity of 95.7% and a delay of 18.2 seconds. Their final test were ran on a graphics card (Nvidia GeForce GTX 1080) which consumes a lot of power. This shows their algorithm have a better sensitivity but a lower specificity and higher delay. One can note that the specificity they obtained was higher for 15 patients but drastically lower for one. The two algorithms thus seem comparable in results but both do not work well for certain patients. Another point is their algorithm is built for low memory usage while the one of this paper was for implementation on FPGA for low consumption.

4 Conclusion

This paper shows that statistical features computed after applying a discrete wavelet transform and linear SVM can be used to detect epileptic seizures for different patients. The input signal is a pre-filtered average EEG over different electrodes and can be used in realtime with the detector. This was demonstrated through different implementations, first more theoretical one to make sure the system worked, and the final implementation was installed on a FPGA board to prove the low-power capability. It was able to detect, in a fast manner, seizures with a high sensitivity and specificity. The results are comparable to techniques which require a high power consumption.

References

[1] Hanneke M. de Boer, Marco Mula and Josemir W Sander. 'The global burden and stigma of epilepsy'. In: *Epilepsy & Behavior*. Current Views on Epilepsy and Behavior 12.4 (1st May 2008), pp. 540–546. ISSN: 1525-5050. DOI: 10.1016/j.yebeh.2007.12.019.

[2] *About epilepsy*. Epilepsy Action. URL: https://www.epilepsy.org.uk/info (visited on 12th Apr. 2023).

[3] Zisheng Zhang and Keshab K. Parhi. 'Low-Complexity Seizure Prediction From iEEG/sEEG Using Spectral Power and Ratios of Spectral Power'. In: *IEEE Transactions on Biomedical Circuits and Systems* 10.3 (June 2016), pp. 693–706. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2015.2477264.

[4] Mustafa Sameer et al. 'Epileptical Seizure Detection: Performance analysis of gamma band in EEG signal Using Short-Time Fourier Transform'. In: *2019 22nd International Symposium on Wireless Personal Multimedia Communications (WPMC)*. Nov. 2019, pp. 1–6. DOI: 10.1109/WPMC48795.2019.9096119.

[5] Kais Gadhoumi, Jean-Marc Lina and Jean Gotman. 'Seizure prediction in patients with mesial temporal lobe epilepsy using EEG measures of state similarity'. In: *Clinical Neurophysiology* 124.9 (1st Sept. 2013), pp. 1745–1754. ISSN: 1388-2457. DOI: 10.1016/j.clinph.2013.04.006.

[6] Oliver Faust et al. 'Wavelet-based EEG processing for computer-aided seizure detection and epilepsy diagnosis'. In: *Seizure* 26 (1st Mar. 2015), pp. 56–64. ISSN: 1059-1311. DOI: 10.1016/j.seizure.2015.01.012.

[7] Abdulhamit Subasi and M. Ismail Gursoy. 'EEG signal classification using PCA, ICA, LDA and support vector machines'. In: *Expert Systems with Applications* 37.12 (1st Dec. 2010), pp. 8659–8666. ISSN: 0957-4174. DOI: 10.1109/TBCAS.2015.2477264.

[8] Shengkun Xie and Sridhar Krishnan. 'Wavelet-based sparse functional linear model with applications to EEGs seizure detection and epilepsy diagnosis'. In: *Medical & Biological Engineering & Computing* 51.1 (1st Feb. 2013), pp. 49–60. ISSN: 1741-0444. DOI: 10.1007/s11517-012-0967-8.

[9] Ning Wang and Michael R. Lyu. 'Extracting and Selecting Distinctive EEG Features for Efficient Epileptic Seizure Prediction'. In: *IEEE Journal of Biomedical and Health Informatics* 19.5 (Sept. 2015), pp. 1648–1659. ISSN: 2168-2208. DOI: 10.1109/JBHI.2014.2358640.

[10] Luigi Chisci et al. 'Real-Time Epileptic Seizure Prediction Using AR Models and Support Vector Machines'. In: *IEEE Transactions on Biomedical Engineering* 57.5 (May 2010), pp. 1124–1132. ISSN: 1558-2531. DOI: 10.1109/TBME.2009.2038990.

[11] Yun Park et al. 'Seizure prediction with spectral power of EEG using cost-sensitive support vector machines'. In: *Epilepsia* 52.10 (2011), pp. 1761–1770. ISSN: 1528-1167. DOI: 10.1111/j.1528-1167.2011.03138.x.

[12] Md Mursalin et al. 'Automated epileptic seizure detection using improved correlation-based feature selection with random forest classifier'. In: *Neurocomputing* 241 (7th June 2017), pp. 204–214. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.02.053.

[13] Elif Derya Übeyli. 'Combined neural network model employing wavelet coefficients for EEG signals classification'. In: *Digital Signal Processing* 19.2 (1st Mar. 2009), pp. 297–308. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2008.07.004.

[14] *The SWEC-ETHZ iEEG Database and Algorithms*. URL: http://ieeg-swez.ethz.ch/ (visited on 9th Mar. 2023).

[15] Uwe Meyer-Baese et al. 'Discrete wavelet transform FPGA design using MatLab/Simulink'. In: *Independent Component Analyses, Wavelets, Unsupervised Smart Sensors, and Neural Networks IV*. Vol. 6247. SPIE, 17th Apr. 2006, pp. 9–18. DOI: 10.1117/12.663457.

[16] Fang Dong et al. 'Novel seizure detection algorithm based on multi-dimension feature selection'. In: *Biomedical Signal Processing and Control* 84 (1st July 2023), p. 104747. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2023.104747.

[17] Ling Guo et al. 'Automatic feature extraction using genetic programming: An application to epileptic EEG classification'. In: *Expert Systems with Applications* 38.8 (1st Aug. 2011), pp. 10425–10436. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2011.02.118.

[18] Andrey A. Efanov, Sergey A. Ivliev and Alexey G. Shagraev. 'Welford's algorithm for weighted statistics'. In: *2021 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE)*, pp. 1–5. DOI: 10.1109/REEPE51337.2021.9387973.

[19] Martin Hirsch, Dirk-Matthias Altenmüller and Andreas Schulze-Bonhage. 'Latencies from intracranial seizure onset to ictal tachycardia: A comparison to surface EEG patterns and other clinical signs'. In: *Epilepsia* 56.10 (2015), pp. 1639–1647. ISSN: 1528-1167. DOI: 10.1111/epi.13117.

[20] Alessio Burrello et al. 'One-shot Learning for iEEG Seizure Detection Using End-to-end Binary Operations: Local Binary Patterns with Hyperdimensional Computing'. In: *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2018, pp. 475–478. ISBN: 978-1-5386-3603-9. DOI: 10.1109/BIOCAS.2018.8584751.

Appendices

A Table of the features kept for each patient

Table 1: The name of the features indicates from which sub-bands it was computed and which statistics it is, where ene is energy, std is standard deviation, cur is curve length, max is maximum and min is minimum.

ID	Features kept
1	D1_ene, D2_ene, D3_ene, D4_ene, D1_std, D2_std, D1_cur, D2_cur, Signal_ene
2	D2_ene, Signal_ene
3	D1_cur
4	D3_ene, D5_ene, D6_ene, D5_cur, D6_cur, A6_cur, D2_max, Signal_std, Signal_max, Signal_min
5	D5_ene, D6_ene, D1_std, Signal_ene
6	D2_ene, D3_ene, D4_ene, D3_std, D3_cur
7	D2_ene, D3_ene
8	D1_ene, D1_std
9	D3_ene, D4_ene, D5_ene, D3_std, D3_max, D4_max, Signal_std, Signal_ene
10	D1_ene, D2_ene, D3_ene, D2_std, D1_cur, D2_cur, Signal_cur
11	D3_ene, D4_ene, D5_ene, D4_cur, Signal_std, Signal_cur
12	D2_ene, D3_ene, D2_std, D3_std, D2_max, D3_max, Signal_std, Signal_ene
13	D2_ene, D3_ene, D5_ene, D2_cur, D3_cur, D4_cur, Signal_std, Signal_ene, Signal_cur
14	D3_ene, D4_ene, D5_ene, D6_ene, A6_ene, D6_std, A6_cur, Signal_std, Signal_ene, Signal_max
15	D5_ene, D5_max, Signal_std, Signal_ene, Signal_max
16	D1_ene, D2_ene, D3_ene, D1_std, D1_cur, Signal_std, Signal_ene, Signal_cur

B Table of the miss-classification costs

Table 2: Miss-classification costs kept for each patient. C^+ is associated with ictal epochs, meaning a high value will result in less false negative. C^- is the opposite and will result in less false positive.

ID	C^+	C^-
1	3	1
2	2	1
3	2	1
4	12	5
5	2	1
6	3	1

Continued on next page

ID	C^+	C^-
7	12	1
8	3	1
9	2	1
10	15	1
11	2	1
12	4	1
13	3	1
14	1	2
15	3	1
16	3	1

C Example of generated VHDL code for the wavelet transform of level 1

```
-- -------------------------------------------------------------
--
-- File Name: hdlsrc\full_system\alpha1_level_dwt.vhd
-- Module: alpha1_level_dwt
--
-- Generated by MATLAB 9.11 and HDL Coder 3.19
--
-- -------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.full_system_pkg.ALL;

ENTITY alpha1_level_dwt IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        enb_1_2_0                         :   IN    std_logic;
        enb_1_2_1                         :   IN    std_logic;
        s_in                              :   IN    std_logic_vector(31 DOWNTO 0);  -- sfix32_En12
        enable                            :   IN    std_logic;
        D                                 :   OUT   std_logic_vector(50 DOWNTO 0);  -- sfix51_En12
        A                                 :   OUT   std_logic_vector(50 DOWNTO 0)  -- sfix51_En12
        );
END alpha1_level_dwt;


ARCHITECTURE rtl OF alpha1_level_dwt IS

  -- Component Declarations
  COMPONENT Discrete_FIR_Filter
    PORT( clk                             :   IN    std_logic;
          reset                           :   IN    std_logic;
          enb                             :   IN    std_logic;
          Discrete_FIR_Filter_in          :   IN    std_logic_vector(31 DOWNTO 0);  -- sfix32_En12
          Discrete_FIR_Filter_coeff       :   IN    vector_of_std_logic_vector16(0 TO 7);  -- int16 [8]
          Discrete_FIR_Filter_out         :   OUT   std_logic_vector(50 DOWNTO 0)  -- sfix51_En12
          );
  END COMPONENT;

  COMPONENT Discrete_FIR_Filter1
    PORT( clk                             :   IN    std_logic;
          reset                           :   IN    std_logic;
          enb                             :   IN    std_logic;
          Discrete_FIR_Filter1_in         :   IN    std_logic_vector(31 DOWNTO 0);  -- sfix32_En12
          Discrete_FIR_Filter1_coeff      :   IN    vector_of_std_logic_vector16(0 TO 7);  -- int16 [8]
```

```
48          Discrete_FIR_Filter1_out        :   OUT   std_logic_vector(50 DOWNTO 0)  -- sfix51_En12
49          );
50  END COMPONENT;
51
52  -- Component Configuration Statements
53  FOR ALL : Discrete_FIR_Filter
54    USE ENTITY work.Discrete_FIR_Filter(rtl);
55
56  FOR ALL : Discrete_FIR_Filter1
57    USE ENTITY work.Discrete_FIR_Filter1(rtl);
58
59  -- Signals
60  SIGNAL RT2_bypass_reg                   : std_logic;  -- ufix1
61  SIGNAL RT2_out1                         : std_logic;
62  SIGNAL switch_compare_1                 : std_logic;
63  SIGNAL Constant_out1                    : signed(50 DOWNTO 0);  -- sfix51_En12
64  SIGNAL db4_wavelet_out1                 : vector_of_signed16(0 TO 7);  -- int16 [8]
65  SIGNAL db4_wavelet_out1_1               : vector_of_std_logic_vector16(0 TO 7);  -- ufix16 [8]
66  SIGNAL Discrete_FIR_Filter_out1         : std_logic_vector(50 DOWNTO 0);  -- ufix51
67  SIGNAL Discrete_FIR_Filter_out1_signed  : signed(50 DOWNTO 0);  -- sfix51_En12
68  SIGNAL Downsample_out1                  : signed(50 DOWNTO 0);  -- sfix51_En12
69  SIGNAL Switch_out1                      : signed(50 DOWNTO 0);  -- sfix51_En12
70  SIGNAL switch_compare_1_1               : std_logic;
71  SIGNAL db4_scaling_out1                 : vector_of_signed16(0 TO 7);  -- int16 [8]
72  SIGNAL db4_scaling_out1_1               : vector_of_std_logic_vector16(0 TO 7);  -- ufix16 [8]
73  SIGNAL Discrete_FIR_Filter1_out1        : std_logic_vector(50 DOWNTO 0);  -- ufix51
74  SIGNAL Discrete_FIR_Filter1_out1_signed : signed(50 DOWNTO 0);  -- sfix51_En12
75  SIGNAL Downsample1_out1                 : signed(50 DOWNTO 0);  -- sfix51_En12
76  SIGNAL Switch1_out1                     : signed(50 DOWNTO 0);  -- sfix51_En12
77
78  BEGIN
79    -- FIR Filter: 'db4_wavelet'
80    u_Discrete_FIR_Filter : Discrete_FIR_Filter
81      PORT MAP( clk => clk,
82                reset => reset,
83                enb => enb,
84                Discrete_FIR_Filter_in => s_in,  -- sfix32_En12
85                Discrete_FIR_Filter_coeff => db4_wavelet_out1_1,  -- int16 [8]
86                Discrete_FIR_Filter_out => Discrete_FIR_Filter_out1  -- sfix51_En12
87                );
88
89    -- FIR Filter: 'db4_scaling'
90    u_Discrete_FIR_Filter1 : Discrete_FIR_Filter1
91      PORT MAP( clk => clk,
92                reset => reset,
93                enb => enb,
94                Discrete_FIR_Filter1_in => s_in,  -- sfix32_En12
95                Discrete_FIR_Filter1_coeff => db4_scaling_out1_1,  -- int16 [8]
96                Discrete_FIR_Filter1_out => Discrete_FIR_Filter1_out1  -- sfix51_En12
97                );
98
99    -- Dowsample the enable signal
100   RT2_bypass_process : PROCESS (clk, reset)
101   BEGIN
102     IF reset = '1' THEN
103       RT2_bypass_reg <= '0';
104     ELSIF clk'EVENT AND clk = '1' THEN
105       IF enb_1_2_1 = '1' THEN
106         RT2_bypass_reg <= enable;
107       END IF;
108     END IF;
109   END PROCESS RT2_bypass_process;
110   RT2_out1 <= enable WHEN enb_1_2_1 = '1' ELSE
111       RT2_bypass_reg;
112
113   -- Used to know if the enable signal is high
114   switch_compare_1 <= '1' WHEN RT2_out1 > '0' ELSE
115       '0';
116
117   -- Constant 0 at output if the enable signal is low
118   Constant_out1 <= to_signed(0, 51);
119
120   -- Filter values for db4_wavelet
121   db4_wavelet_out1(0) <= to_signed(16#0000#, 16);
122   db4_wavelet_out1(1) <= to_signed(16#0001#, 16);
123   db4_wavelet_out1(2) <= to_signed(-16#0001#, 16);
```

```
  db4_wavelet_out1(3) <= to_signed(16#0000#, 16);
  db4_wavelet_out1(4) <= to_signed(16#0000#, 16);
  db4_wavelet_out1(5) <= to_signed(16#0000#, 16);
  db4_wavelet_out1(6) <= to_signed(16#0000#, 16);
  db4_wavelet_out1(7) <= to_signed(16#0000#, 16);

  outputgen1: FOR k IN 0 TO 7 GENERATE
    db4_wavelet_out1_1(k) <= std_logic_vector(db4_wavelet_out1(k));
  END GENERATE;
  -- Connect the output of the db4_wavelet filter to the switch
  Discrete_FIR_Filter_out1_signed <= signed(Discrete_FIR_Filter_out1);

  -- Downsample output register
  Downsample_output_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Downsample_out1 <= to_signed(0, 51);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb_1_2_0 = '1' THEN
        Downsample_out1 <= Discrete_FIR_Filter_out1_signed;
      END IF;
    END IF;
  END PROCESS Downsample_output_process;

  -- Choose between the downsampled output and the constant 0
  Switch_out1 <= Constant_out1 WHEN switch_compare_1 = '0' ELSE
      Downsample_out1;

  D <= std_logic_vector(Switch_out1);

  -- Same as above but for the db4_scaling filter
  switch_compare_1_1 <= '1' WHEN RT2_out1 > '0' ELSE
       '0';

  -- Filter values for db4_scaling
  db4_scaling_out1(0) <= to_signed(16#0000#, 16);
  db4_scaling_out1(1) <= to_signed(16#0000#, 16);
  db4_scaling_out1(2) <= to_signed(16#0000#, 16);
  db4_scaling_out1(3) <= to_signed(16#0000#, 16);
  db4_scaling_out1(4) <= to_signed(16#0000#, 16);
  db4_scaling_out1(5) <= to_signed(16#0001#, 16);
  db4_scaling_out1(6) <= to_signed(16#0001#, 16);
  db4_scaling_out1(7) <= to_signed(16#0000#, 16);

  outputgen: FOR k IN 0 TO 7 GENERATE
    db4_scaling_out1_1(k) <= std_logic_vector(db4_scaling_out1(k));
  END GENERATE;

  Discrete_FIR_Filter1_out1_signed <= signed(Discrete_FIR_Filter1_out1);

  -- Downsample output register
  Downsample1_output_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Downsample1_out1 <= to_signed(0, 51);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb_1_2_0 = '1' THEN
        Downsample1_out1 <= Discrete_FIR_Filter1_out1_signed;
      END IF;
    END IF;
  END PROCESS Downsample1_output_process;

  Switch1_out1 <= Constant_out1 WHEN switch_compare_1_1 = '0' ELSE
      Downsample1_out1;

  A <= std_logic_vector(Switch1_out1);

END rtl;
```

D Finite state machines for the interface and the Nios II CPU

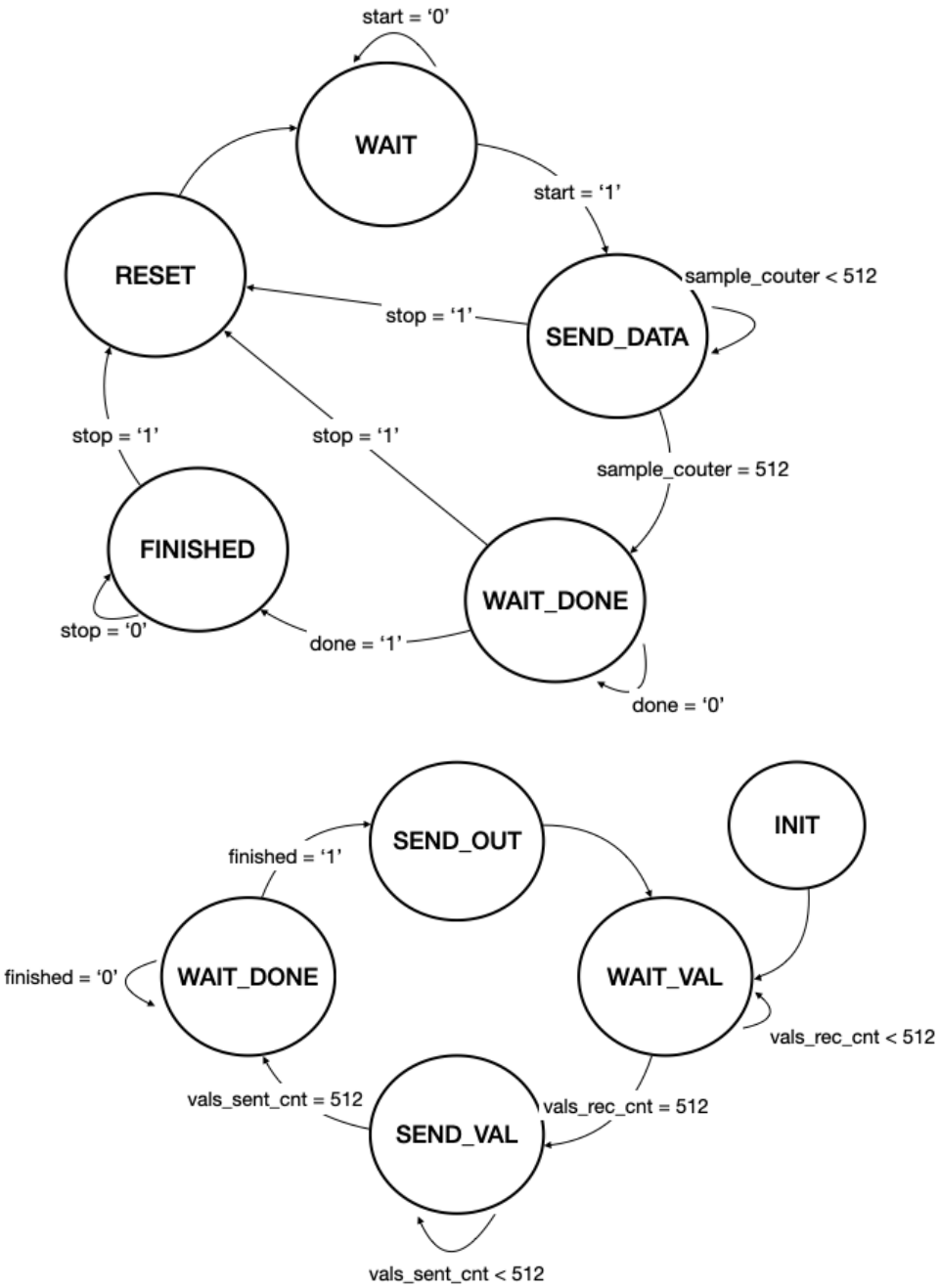

Figure D.0: Top: the FSM for the VHDL interface to the predictor. The signals **start** and **stop** are written by the Nios II CPU. The **sample_counter** is an internal register. The signal **done** is read from the predictor.
Bottom: the FSM for the Nios II CPU. The two counters are in the C code, the signal **finished** is polled from the interface.

E VHDL code of the interface and C code for the Nios II CPU

Listing 1: Example of part of the FSM process. Here it sends one by one each value to current_sample which is connected to the signal_in of the predictor.

```
-- Main FSM
process(clk)
begin
  if rising_edge(clk) then
    case state is
    when SEND_DATA =>
      -- Send data to the SVM, there are 512 samples
      if stop = '1' then
        enable <= '0';
        state <= RESET;
      else
        enable <= '1';
```

```
        if sample_counter = 512 then
          current_sample <= (others => '0');
          state <= WAIT_DONE;
        else
          current_sample <= samples(to_integer(sample_counter));
          sample_counter <= sample_counter + 1;
          state <= SEND_DATA;
      end if;
    end if;
    ...
  end if;
end process;
```

Listing 2: Example of the C code ran by the Nios II CPU. Here one can see how data are received from MATLAB and then sent to the interface. The register map of the interface is in table 3

```
-- In main loop
switch (state) {
case WAIT_VAL:
  // Check if there is 2 values in the RS-232 FIFO (LSB and MSB)
  if (alt_up_rs232_get_used_space_in_read_FIFO(rs232_device) >= 2) {
    // Read the 2 bytes from the RS-232 FIFO
    // because RS-232 send data as 8-bit packets
    alt_up_rs232_read_data(rs232_device, data, NULL);
    uint8_t lsb = *data;
    alt_up_rs232_read_data(rs232_device, data, NULL);
    uint8_t msb = *data;

    // Convert the 2 bytes to a 16-bit value
    uint16_t value = (msb << 8) | lsb;

    // Write the value to an internal FIFO
    fifo_write(&fifo, value);

    // If we stored 512 values in the FIFO, go to send value state
    if (fifo_num_elements(&fifo) >= 512) {
      state = SEND_VAL;
      printf("Received 512 vals\n");
    }
  }

  break;

case SEND_VAL:
  // Writes each of the 512 values to the predictor
  for (int i = 0; i < 512; i++) {
    // Read the next value from the FIFO
    uint16_t value = fifo_read(&fifo);

    // Write the value to the predictor (the addresses are (0 to 511) * 4
    // because we send a value of 4 bytes at a time)
    IOWR_32DIRECT(PREDICTOR_INTERFACE_0_BASE, 4 * i, (uint32_t) value);
  }

  // Send order of starting predictor
  IOWR_32DIRECT(PREDICTOR_INTERFACE_0_BASE, 4 * START_ADDR, 1);

  // Go to wait for done state
  state = WAIT_DONE;
  printf("Sent vals to pred, wait for done\n");

  // Disable start signal to predictor
  IOWR_32DIRECT(PREDICTOR_INTERFACE_0_BASE, 4 * START_ADDR, 0);

  break;

...
}
```

Table 3: This shows the addresses of the internal registers of the predictor interface which are accessible through the Avalon slave. Note that registers which have more than 32 bits require 2 addresses (they all have 42 bits), the first one being the LSB. Finally, the number of registers depend on the number of features required, where $n = 6i + 519$ with i the number of features.

Address	Register	Function
0	samples(0)	First sample of the epoch
	...	
511	samples(511)	Last sample of the epoch
512	start	Indicating when the interface should start sending data to the predictor.
513	stop	Indicating when the interface should reset.
514	finished	Indicate when the output is ready to be read.
515-516	result	The output from the predictor.
517-518	bias	Parameter b for the SVM.
519-520	scale	Parameter *scale* for the SVM.
521-522	beta_first_fea	Parameter β of the first feature of this patient for the SVM.
523-524	mu_first_fea	Parameter μ of the first feature of this patient for the SVM.
525-526	sigma_first_fea	Parameter σ of the first feature of this patient for the SVM.
527-528	beta_sec_fea	Parameter β of the second feature of this patient for the SVM.
	...	
n-(n+1)	sigma_last_fea	Parameter σ of the last feature of this patient for the SVM.