A Chisel-Based DSP Backend and Classifier Generator for Biomedical Signal Feature Extraction and Classification Applications

Adelson Chua, Justin Doong, Ryan Kaveh, Cem Yalcin, Rachel Zoll
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, Berkeley, CA, USA
Email: {adelson.chua, jbdoong, ryankaveh, cemyalcin, rachelzoll}@berkeley.edu

***Abstract*—While neural and physiological monitoring (*e.g.*, EEG, ECG) allow doctors to provide early disease diagnosis, many examinations are *only* performed in the doctor's office and miss slow-changing or sporadic symptoms that span days, thus necessitating the need for unobtrusive and low-power devices that can be comfortably worn during daily life. Such low-power diagnostic devices exist, however given the variety of bio-signals and large number of hardware alternatives to implement bio-signal feature extraction circuits, traditional design methodologies fail to exhaustively explore the design space to find the optimum hardware given area, power, and diagnosis accuracy constraints. A hardware-generator-based approach could effectively explore the design space and benchmark the generated design in order to find the desired type and number of features, as well as the hardware microarchitecture. In this work, we have developed a wellness monitor generator in Chisel that takes desired bio-signal features as high-level specifications, generates appropriate DSP datapaths and a support vector machine (SVM) classifier to meet the specifications, and connects this accelerator to a RISC-V Rocket core. Using this generator, we have demonstrated that it is possible to generate hardware that functionally matches a high-level Python based seizure detector and achieve identical seizure classification.**

I. Introduction

Over the last decade, biomedical-oriented software libraries have become increasingly prevalent. With the Python genomics library [1] and MATLAB's EEGLAB [2] and microbiology cell analysis tools [3], researchers now have the necessary software tools to process biomedical datasets [4]. However, these software tools are relatively slow [5] and power inefficient [4] when compared to their respective hardware methods. For example, hardware accelerators for genomic sequencing can achieve orders of magnitude increases in performance over software solutions [6], while seizure detection co-processors can save up to 20x power while drastically reducing area and latency of seizure detection [7]. While these hardware solutions exist, many biological sciences researchers lack the tools and relevant circuit and architecture expertise necessary to design their own hardware accelerators, and as a result, cannot easily harness accelerator solutions for niche bioscience problems such as sequence analysis, phylogenetics, structural biology, and others [8]. To directly alleviate this need for biomedical-targeted hardware accelerators, we present an easy to use, flexible framework to create a hardware accelerator meant for general health monitoring and biosignal DSP. This generator is built in Chisel that can be integrated in automated design flows (like the one developed for this project - Figure 1). Using a labeled training data set and high level parameters, a MATLAB/Python script can eliminate highly correlated features and channels in order to select the primary features. These features can then be passed to a hardware generator for synthesis, simulation and benchmarking. Depending on results, the flow can iterate on this design to switch out, increase, or decrease the features for a specific design.

A self-sufficient hardware generator is required to make this closed loop exploration from raw signal to finished coprocessor. To this effect, the presented wellness monitor generator can take high level features as an input, our wellness monitor framework generates the appropriate DSP blocks (using a library of generators), wires them appropriately, integrates them with a principle component analysis (PCA) block and a support vector machine (SVM), and lastly connects the generated accelerator to Rocket, a RISC-V based processor, for configuration and testing. A sample application of seizure detection was targeted for this project to motivate and demonstrate the framework's capability to target any type of wellness monitoring.

II. Architecture

The architecture of the wellness monitor was designed to be as extensible (to various different bio-interfaces) and scalable (in number of channels/features) as possible. Therefore, we have constructed a two-part system. First, a Scala-based generator parses high level feature requests and generates parameters for a datapath. This datapath is capable of extracting the requested feature

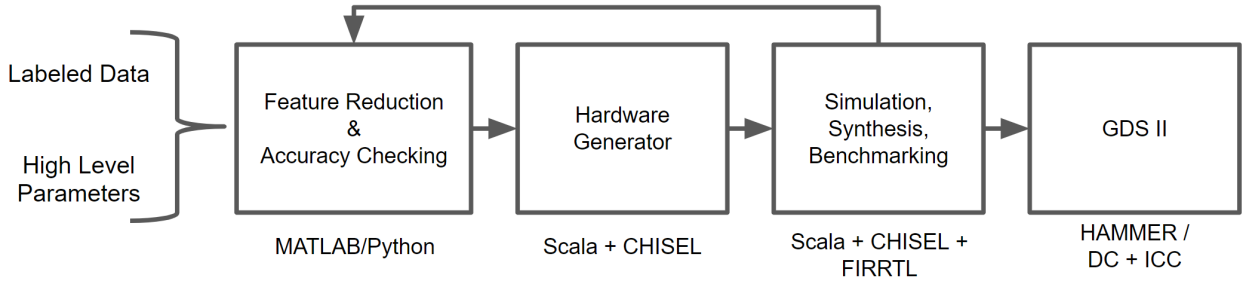

Fig. 1. An automated design flow for a wellness monitor.

from raw physiological data. The second part of the wellness generator takes these parameters, generates the datapaths using a pre-built Chisel library, and tiles them in parallel between the input node and the classifier. Each datapath represents the extraction of a feature from a bio-signal. These features are fed to a PCA which performs dimensionality reduction, eliminating correlation between existing features and reducing the number of features processed by SVM. The newly created features are fed to the SVM to perform classification. The key insight is that each datapath shares the same input and output characteristics. This allows for a regular tiling of features datapaths, each of which perform different calculations. Figure 2 shows a typical wellness monitor generated using the wellness monitor generator.

The generated wellness monitor is connected to a RISC-V Rocket core through a TileLink2 interface. Rocket core is mainly used to feed configuration data (PCA and SVM matrices) to the monitor and read the current classification state of the SVM. The Rocket core also has access to a memory-mapped test stream input of the wellness monitor to test and train the system.

III. Implementation

The wellness monitor generator was implemented primarily in Chisel, an open-source hardware construction language developed at UC Berkeley [9]. The core Chisel code is wrapped in a Scala parser which takes direct feature requests (*e.g.*, 'alpha band spectral bandpower' or 'line length'), generates parameters and a datapath to extract these features, and ultimately configures the Chisel framework to generate these datapaths. In order to motivate and demonstrate the framework's feasibility and flexibility, seizure related features (specifically frequency band power and line length) were chosen for implementation. Lastly, Scala, Python and C models were constructed to verify the generated hardware's output when fed previously recorded neurological data.

A. Wellness Generator

The parser works as a series of Scala functions defined for each supported feature. As the user calls each function, a datapath is defined for the specified channel (*e.g.*, *FIR, FFT Buffer, FFT, Spectral Bandpower*) and its parameters are appended to a collection that keeps track of all requested hardware. Once all datapath parameters are defined, they are passed to the Chisel module `wellnessGen` for actual hardware generation. From here the Chisel module steps through the datapaths, instantiates the appropriate modules (with the given parameters), and stores them in a user readable data structure. Once all modules are instantiated, the generator then connects all blocks' inputs and outputs as specified by the datapath. For the previously defined datapath, this would require driving the *FIR's* input with the ch0 stream, driving the *FFT Buffer* with the FIR output, etc. Once the entire coprocessor has been instantiated and wired, the wellness generator then memory maps its I/O to Rocket core, in addition to generating the appropriate external inputs such that it can stream inputs directly from an ADC.

B. Chisel Modules

A number of building blocks were implemented as part of the sample coprocessor. FIR and IIR filter generators were both designed in order to provide users with flexibility when defining input conditioning and feature filters. Furthermore, both generators are excellent examples of Chisel's type generic functionality.

1) Feature Extraction: A fast fourier transform (FFT) generator was already implemented in Chisel [10] and used to drive frequency domain datapath calculations in the wellness monitor. Since the FFT generator assumes an entirely parallel input, a serial-to-parallel buffer was implemented with a shift register to make the FFT a streaming FFT with an overlap of $n-1$ (where n is the number of bins).

The spectral bandpower and line length, while small modules, are a major part of seizure detection which was determined to be one example target application of the wellness monitor.

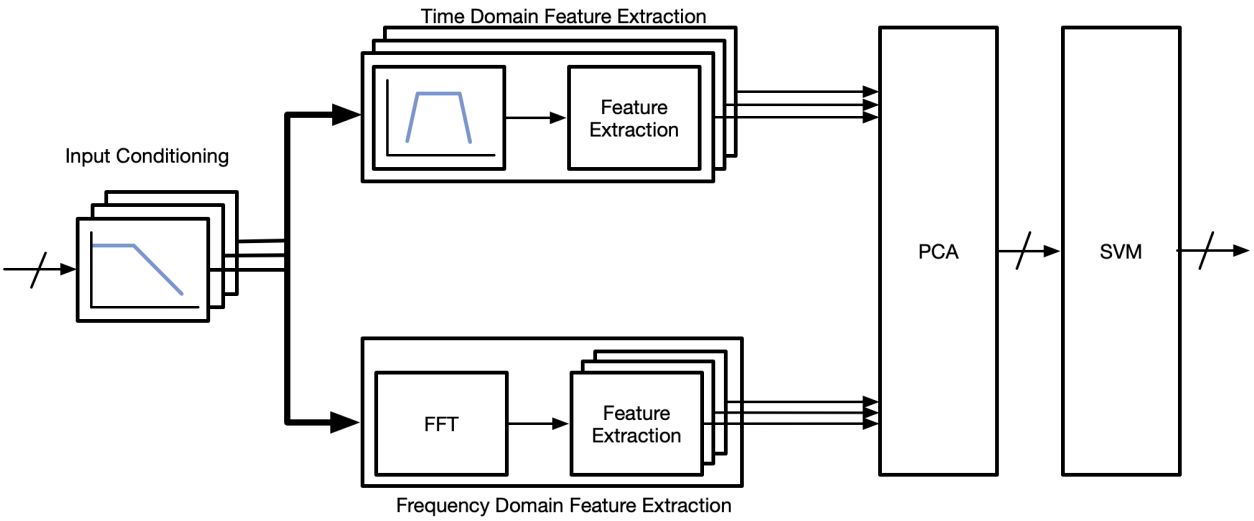

Fig. 2. Top level block diagram template of a sample Wellness Monitor that can be generated using the wellness monitor generator. Multiple channels are streamed to various conditioning blocks, which are then fed to parallel feature extraction *datapaths*. Post-processed data is then fed into a PCA block for dimensionality reduction and then lastly to the SVM for classification.

Spectral bandpower is described as the average power in a given frequency range:

$$\sum \left(\frac{|x(f)|}{N} \right)^2 \tag{1}$$

In order to make the bandpower generator as simple to use as possible, it takes in the entire FFT output vector and calculates only the average power between a start and stop bin (unused bins will be removed during optimization). This allows the input/output characteristics of each bandpower block to be exactly the same and easier for the wellness generator to connect. Examples of commonly used bandpowers include the bandpower of the alpha band (8-15 Hz) and the bandpower of the delta band ($<$ 4 Hz). These bands are quite useful in many EEG applications, including sleep studies and seizure detection.

Line length is formally the summed distance between two adjacent samples over a time window. For seizure detection dataset, line length has been proven to be one of the most significant features that can properly detect such events.

$$\sum |x(i) - x(i-1)| \tag{2}$$

The line length generator does exactly this but also keeps a rolling average of the last *n* calculated line lengths. Where *n* is determined by the block's *window size* parameter. This rolling average is useful in reducing the impact of glitches in the input stream.

2) Dimensionality Reduction: The principal component analysis (PCA) block performs dimensionality reduction of the dataset. This algorithm enables the transformation of the current feature space into a reduced dimension space that captures most of the variance in the original data, via eigendecomposition. Eigendecomposition and the derivation of the principal components are meant to be performed offline due to computational complexity. Only the translation from the original feature space to the reduced dimension space is performed by the hardware accelerator. This translation consists of a matrix-vector multiplication of the data matrix with a vector of feature weights.

3) Classification: The support vector machine (SVM) block performs classification of different classes using the PCA-reduced features. After computing a separating hyperplane between unique classes, points are classified by checking which part of the line the new point belongs to. The actual computation of the decision function, as shown below, is done by the hardware accelerator.

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + b \tag{3}$$

The α_i refers to the support vector weights; $y^{(i)}$ corresponds to the support vector class (either +1 or -1); $x^{(i)}$ would be the support vectors while x refers to the new input vector; b corresponds to the intercept. These terms are summed up for every feature m supported by the SVM. All of these parameters are generated from offline training.

For non-linearly separable classes, a kernel trick is employed to transform the data projection into another dimension, allowing the separating hyperplane to curve in the lower dimensional space. The hardware generator can either calculate polynomial kernels of varying degrees and the radial basis function kernels.

Multi-class classification is done by instantiating multiple binary classifiers where the final class is determined through voting. There are three general approaches for multi-class classification used by common machine learning libraries: One vs Rest, One vs One, and Error-correcting Output Codes. All of these are supported by the Chisel-based generator.

Both the PCA and SVM blocks require several reference matrices (feature normalization and transformation matrices for PCA, support vector and weight matrices and intercept vector for SVM) to perform their computations. These matrices are stored in a register array that is instantiated alongside the generated wellness monitor This memory is configured by the Rocket chip through the TileLink2 interface and is meant to be preloaded with values in the application layer, after offline training using a machine learning software library has been performed.

C. Scala Tester

In order to confirm the correct functionality of all of described generators, Scala golden models and unit tests were written for each module type. For a given unit Chisel module, a tester would instantiate both the golden model (a functionally equivalent, software version of the design under test (DUT)) and the DUT. It then feeds random data of a given data type (unsigned/signed integer or fixed point) to both the golden model and the DUT and compares their outputs.

These unit golden models were also used to test the functionality of entire generated wellness monitor as well. When testing, the wellness generator parser mentioned earlier not only generates Chisel parameters for a hardware version of requested datapaths, but golden model parameters as well. These golden model parameters are used to generate a Scala version of the requested set of datapaths using the original unit golden models. The tester then provides the same stimulus to both the Chisel monitor and the Scala golden model monitor and compares their outputs both for signed integer and fixed point implementations. It is important to emphasize that both the Chisel and Scala implementations of wellness generator take in arbitrary parameters and can build the appropriate datapaths for those parameters.

D. Application-specific Comprehensive Test Setup

A test framework was developed to enable application-specific testing given some raw data that needs to be classified. A Python script performs SVM training from the raw data while also calculating the expected accuracies after the model has been created. All parameters, configuration matrices, and input vectors are then written to different files to configure the Scala-based top-level tester. The tester sets all submodule generator parameters accordingly and passes an actual input from the test dataset through the datapath chain. Afterwards, the expected values from the SVM classifier and the configuration matrices are written to a C header file. This will then be used in the integration test where the wellness datapath is integrated with a RISC-V core. This checks that the expected output from the Python model are consistent and are synthesizeable in hardware.

IV. Results

A sample seizure detection setup was implemented to test the wellness monitor hardware generator. Figure 3 shows the waveform of the labels calculated by the hardware generator on top of the EEG signal that contains both non-seizure and seizure signatures. In this configuration, the 500 Hz sample input signal is passed through an FIR filter with a 200 Hz cut-off. Three features were then calculated: line length and alpha (8-15 Hz) and theta (4-7 Hz) spectral bandpowers. These then go to the PCA block which reduces the three features down to a single feature to be used by the SVM block for classification. In this setup, the generator classified the dataset with a sensitivity (true positive rate) of 93.7% and a specificity (true negative rate) of 88.5%. These results are consistent (within 10% accurate; accuracy limitation is due to fixed point calculations) with the Python model that was developed, which replicates the calculations done by the accelerator, demonstrating the correctness of the generator and synthesizeability of the design.

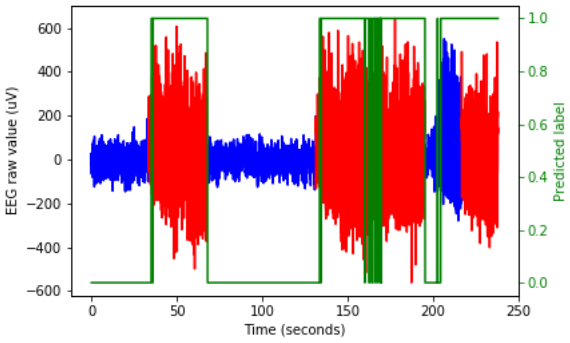

Fig. 3. Seizure detection results from the wellness monitor generator. Red signals refer to seizure events while blue corresponds to non-seizure events.

V. Conclusion

In this work, an extensible and scalable wellness monitor generator was implemented in Chisel. The architecture of our wellness monitor generator makes it easy to include new health feature extractors. The generator automatically creates the collection of desired features, decides on the datapath that computes the feature, organizes datapaths into a human readable data-structure,

generates modules, and automatically wires modules and a RISC-V core together. Hence, while we did not implement an entire library of feature extractors, it should be relatively easy for other users to write their own modules and include them in the top-level datapath. While PCA and SVM training are currently done offline, a vector processor could enable online training. Moreover, better and more complex classifiers can even be implemented to be integrated with the generator, such as neural nets.

Concretely, this architecture provides a processor and datapath platform which enables painless prototyping of individual hardware blocks, as well as exhaustive algorithm-hardware design space exploration.

VI. Future Work

Currently, the generated datapaths cannot share hardware in a generalized way (*e.g.*, each *Bandpower* feature invokes a new FFT instead of reusing the same block). One possible solution is to store the different datapaths into a tree-like structure so that modules can be connected the same way the tree is traversed from root to leaf. Furthermore, more feature models need to be designed and added to the wellness library. One additional feature that is being explored right now is the discrete wavelet transform which captures both frequency and temporal information about the signal. This is used to perform QRS detection in electrocardiograms and has been shown to be capable of detecting tachycardia and bradycardia. Lastly, a graphical user interface could greatly improve a user's quality of life when designing their own wellness monitor.

References

[1] S. P. Ong, W. D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V. L. Chevrier, K. A. Persson, and G. Ceder, "Python materials genomics (pymatgen): A robust, open-source python library for materials analysis," *Computational Materials Science*, vol. 68, pp. 314 – 319, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0927025612006295

[2] A. Delorme and S. Makeig, "Eeglab: an open source toolbox for analysis of single-trial eeg dynamics including independent component analysis," *Journal of Neuroscience Methods*, vol. 134, no. 1, pp. 9 – 21, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165027003003479

[3] B. Obara, M. A. Roberts, J. P. Armitage, and V. Grau, "Bacterial cell identification in differential interference contrast microscopy images," *BMC Bioinformatics*, vol. 14, no. 1, p. 134, Apr 2013. [Online]. Available: https://doi.org/10.1186/1471-2105-14-134

[4] A. Page, N. Attaran, C. Shea, H. Homayoun, and T. Mohsenin, "Low-power manycore accelerator for personalized biomedical applications," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '16. New York, NY, USA: ACM, 2016, pp. 63–68. [Online]. Available: http://doi.acm.org/10.1145/2902961.2902986

[5] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," *SIGPLAN Not.*, vol. 53, no. 2, pp. 199–213, Mar. 2018. [Online]. Available: http://doi.acm.org/10.1145/3296957.3173193

[6] J. Chiang, M. Studniberg, J. Shaw, S. Seto, and K. Truong, "Hardware accelerator for genomic sequence alignment," *Annu. Int. Conf. IEEE Eng. Med. Biol. - Proc.*, no. Fig 1, pp. 5787–5789, 2006.

[7] A. Jafari, A. Page, C. Sagedy, E. Smith, and T. Mohsenin, "A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix," in *2015 IEEE Biomed. Circuits Syst. Conf.* IEEE, oct 2015, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7348376/

[8] T. Majumder, P. P. Pande, and A. Kalyanaraman, "Hardware Accelerators in Computational Biology: Application, Potential, and Challenges," *IEEE Des. Test*, vol. 31, no. 1, pp. 8–18, feb 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6657749/

[9] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Aviienis, J. Wawrzynek, and K. Asanovi, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[10] A. Wang, J. Bachrach, and B. Nikoli, "A generator of memory-based, runtime-reconfigurable 2n3m5kfft engines," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, pp. 1016–1020.