Foresee: A Modular and Open Framework to Explore Integrated Processing on Brain-Computer Interfaces

Aasmaan Yadav[1], Fernando Cuello Garcia[2*], Alejandro Gonzalez[2*], Bernardo Eilert Trevisan[2*], Anna Xu[2], Muhammed Ugur[2], Abhishek Bhattacharjee[2], Raghavendra Pradyumna Pothukuchi[2†]

[1]Duke University [2]Yale University

***Abstract*— Brain-computer interfaces (BCIs) with processing integrated on the device enable fast and autonomous closed-loop interaction with the brain. While such BCIs are rapidly gaining traction, they are also difficult to design due to the tight and conflicting power and performance needs of on-device processing. Meeting these specifications often requires the BCI processors to be co-designed with applications and algorithms, with processor designers and computational neuroscientists working closely to converge on the target hardware platform. But, this process has traditionally been cumbersome and ad hoc, due to the lack of systematic design space exploration frameworks. In response, we present *Foresee*, a new framework for fast exploration of BCI processors. *Foresee* offers a unified and modular interface for iteratively co-optimizing BCI processors with their algorithms, without sacrificing accuracy, speed, or ease of use. *Foresee* is publicly available, and comes with a library of hardware blocks for common signal processing functions that the community could contribute and build on. We demonstrate *Foresee*'s utility and capability by analyzing on-device processing for two seizure detection methods from prior work, and validating our analysis on real hardware. We expect *Foresee* to be vital in designing next-generation BCIs.**

I. Introduction

Brain-Computer Interfaces (BCIs) connect neural activity in the brain with computers and machines, enabling novel ways to shed light on brain function and disease, restore lost function, treat dysfunction, and more [18]. Wearable and implanted BCIs have evolved from proof-of-concept demonstrations to viable solutions for research and therapeutics [39].

Recent years have witnessed a move towards integrating *on-device processing* on BCIs [13, 16, 18, 28]. Unlike most existing BCIs, which offload processing to servers external to the brain, BCIs with on-device processing offer sub-millisecond responsiveness and autonomous operation without relying on external connections and tethering [16, 28, 29].

Unfortunately, developing on-device processors for BCIs (especially those that are implanted) is challenging. This is because BCI processors must aggressively constrain power consumption and thermal elevation to ensure safe operation, while simultaneously meeting the high computational demands of emerging applications. These constraints preclude typical low-power microcontrollers [4], low-power Graphics Processing Units (GPUs) from mobile devices [21], or similar off-the-shelf designs, as they cannot meet the safe power and temperature while offering adequate compute capability.

Consequently, BCI processors are developed by *co-design*—specializing the processing hardware to the applications that the BCI is expected to run, while also modifying the computational methods to fit on resource-constrained processors [22, 28, 30, 31]. This is an iterative methodology where processor designers and computational neuroscientists explore the suitability of various processor and algorithmic optimizations, validate them on their datasets, develop prototypes and test them, before finalizing the design. The promise of co-design is that it achieves superior performance per watt than other design approaches.

But, unfortunately, no existing framework systematically explores BCI processor development and co-design. This leads to several technical and logistical challenges. For example, the tools and programming environments used by computational neuroscientists and processor designers are strikingly different, making it difficult for the two groups to collaborate. Typically, high-level programming languages like Python or MATLAB are used in algorithmic development for their convenience of usage and abstraction over low-level computational details, whereas processor design needs detailed precise specification through hardware description languages like Verilog, VHDL, SystemC, *etc.* This makes exchanging specifications and optimizations between teams cumbersome, ad hoc, and error-prone. Furthermore, given the lengthy timeframe (*i.e.*, several months or more) needed to prototype a processor, validating the processor against the expected results of the algorithm occurs only much later. These prolonged design cycles make it harder to rework optimizations in a timely manner.

We believe that there is an urgent need for a new framework to simplify and accelerate the iterations needed to co-design BCI algorithms with hardware, and validate the co-design rapidly. This framework must simultaneously offer convenient algorithmic development in high-level languages, and the precise specification for hardware description.

To meet this need, this paper presents *Foresee*[1], a new tool for *fast* and *early-stage* exploration of BCI processors. *Foresee* allows users to define neural signal processing pipelines as a chain of "Modules" that can be described in Python during algorithmic development. Modules can also

*Authors contributed equally.

†Corresponding author.

This material is based upon work partly supported by the National Science Foundation under Grant 2127309 to the Computing Research Association for the CIFellows Project awarded to Raghavendra Pradyumna Pothukuchi.

[1]*Foresee* is a fast, open, and reconfigurable simulator for early exploration of processing in BCIs.

be connected to Verilog hardware implementations, when exploring processor design. *Foresee* would then report detailed hardware power consumption and latency estimates from this description. This serves as a *unified platform* for co-design. Importantly, *Foresee* also allows plugging in conventional task-specific tools that users might be familiar with, offering compatibility with existing workflows. Finally, *Foresee* also makes a library of Verilog hardware description blocks for several signal processing methods open-source, enabling the broader community to contribute and build on it[2].

Foresee is useful in several ways. Computational neuroscientists can use *Foresee* to evaluate the suitability of different algorithms and their variants for on-device processing. Together with processor designers, they can also use *Foresee* to co-design algorithms and hardware. Neuroengineers and BCI designers can use *Foresee* to quickly evaluate whether the new processing features they plan to integrate into BCI are feasible from the perspective of hardware latency and power consumption. These feasibility studies are also helpful for regulators evaluating the possibility of integrating critical features (*e.g.*, encryption) on resource-constrained processors.

Overall, *Foresee* is a first-of-its-kind framework to explore integrated processing on BCIs. We demonstrate *Foresee* by using it to specify two seizure detection algorithms from prior work, explore the tradeoffs in their algorithmic accuracy and hardware costs, and finally, validate them on real Field Programmable Gate Array (FPGA) hardware, demonstrating *Foresee*'s utility and capability.

II. Background & Motivation

Both wearable and implantable BCIs are becoming increasingly capable of supporting a rich set of applications. By leaning on advances in neural sensing and computational neural decoding [18, 39], modern BCIs can read and process several tens of megabits per second (Mbps) of neural data [38] with complex signal processing methods to infer intended movement or speech, detect aberrant activity (as in epilepsy and Parkinson's), and respond in seconds [32, 34, 39].

But, BCIs still offload most of their processing to servers away from the brain, or curtail the data rates or functionality they support. These choices restrict BCI responsiveness, performance, or limit their use to the vicinity of servers [29].

Overcoming current limitations and delivering on the promise of BCIs requires additional focus on *on-device processing*. Such processing can offer sub-millisecond response times and autonomous operation without relying on external connections [16, 22, 28, 31], enabling new BCI applications that have not been feasible until now.

A. On-Device BCI Processing

Safe near-brain operation requires power and energy dissipation to be capped in order to prevent overheating of the brain and to prolong device battery life. These constraints are especially stringent for implanted BCIs, which cannot heat brain tissue by more than 1 °C. This limits the power consumption of the BCI processor to a few milliwatts [26, 37]. In contrast, state-of-the-art mobile phone processors typically consume a few watts, which is ≈100× higher [8]. Under these tight constraints, BCI processors must run complex processing required by applications on the growing volumes of neural data that the sensors record, now in tens of Mbps.

Off-the-shelf low-power microcontrollers, Digital Signal Processors (DSPs), or low-power GPUs, and similar processors (*e.g.*, [4, 21]), are designed for more general purpose use, and fail to meet the power or energy efficiency requirements of near-brain BCI processing. This has prompted researchers to pursue more efficient hardware, where features are customized to the BCI algorithms that the device runs. At the same time, the algorithm is also refactored to operate on resource-constrained processors. In the past, such co-design has led to a variety of BCI processors ranging from highly-specialized Application-Specific Integrated Circuits (ASICs) [20, 38] to more flexible designs [22, 28, 30, 31].

B. BCI Processor Co-Design and Infrastructure Requirements

While the need for co-design through iterative algorithm and processor optimization is not unique to BCIs (*e.g.*, [24]), the constraints of near-brain processing pose unique challenges. Foremost, BCIs need *much tighter* development cycles between the algorithm and processor optimization than in other domains (*e.g.*, machine learning and datacenters). This is because BCI algorithms are becoming increasingly complex and data-intensive while the device power consumption and thermal profiles are highly constrained. Without significant co-design, on-device BCI processing will be unable to meet the performance per watt targets needed for near-brain processing [22, 28, 31].

At the same time, BCI algorithm and hardware development focus on different objectives (*e.g.*, ease of design and accuracy in the algorithm versus hardware power consumption), requiring different expertise and dedicated workflows. Yet, they must easily exchange specifications (*e.g.*, channel counts, accuracy, latency, throughput, power) and optimized designs, and validate their performance. This poses a need for a *unified* platform that provides the convenience of languages like Python and its ecosystem in algorithm development, but the precision of Verilog in hardware description, and support for hardware design automation tools.

Importantly, a tool for BCI processor exploration must also offer flexibility, speed, *and* accuracy to be useful. BCI algorithms are evolving rapidly, and a given task (*e.g.*, seizure detection) has several algorithms and variants, each of which work effectively under different conditions [27]. It is necessary to iterate over this *large* and *changing* design space quickly. Historically, computer scientists have developed several tools and simulators that sacrifice accuracy of processor analysis for speed (*e.g.*, [6, 14, 19]). However, existing state-of-the-art tools suffer from inaccuracies in the range of watts of power. This may be acceptable when exploring the design of servers that consume tens to hundreds of watts, but is unsuitable for implantable BCIs, where even a few milliwatts of inaccuracy risks overheating the brain.

[2]Foresee is publicly available at https://github.com/bci-foresee/foresee.

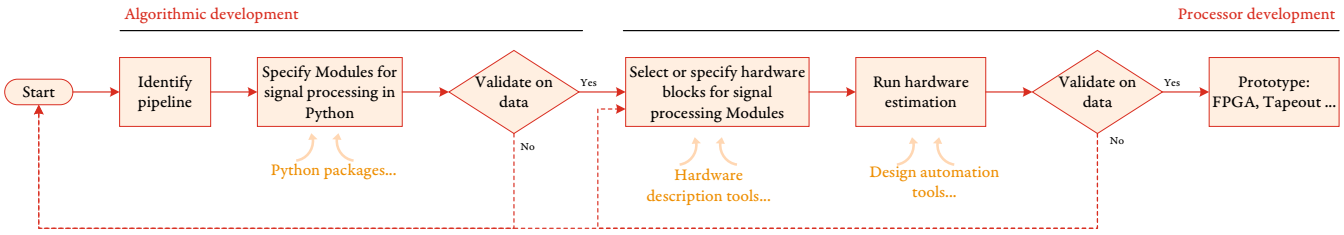


Fig. 1: Overall workflow with *Foresee*. There is a unified interface for processor exploration and co-design with the algorithm. Each task can use corresponding languages, tools, and packages, enabling compatibility with existing toolsets. Hardware blocks can be custom specified or selected from *Foresee*'s open collection.

Similarly, there are some high-level synthesis (HLS) tools (*e.g.* [1]) and frameworks (*e.g.*, PyMTL [5, 15]) that offer precise hardware development and analysis in Python. However, these are intended for use by processor designers and are not easy to use for BCI algorithm development.

What we seek is a new tool, one that offers Python-level convenience for application development, and the accuracy and precision for hardware description and analysis. Today, there exists no unified, fast, and modular framework for exploring BCI processing and co-design.

III. *Foresee*: A New Framework to Explore Processing On-BCI

Figure 1 shows the overall workflow enabled by *Foresee*. The framework facilitates co-design and processor exploration by building neural signal processing pipelines with "Modules" that can be seamlessly specified in Python for algorithmic development or in Verilog for hardware description, and iterate between them with a single interface. *Foresee* also releases an open-source collection of hardware blocks for several signal processing methods that users can utilize directly.

This section describes *Foresee*'s interface and operation. Section IV describes *Foresee* being used to explore hardware design for two published seizure detection methods.

A. Graphical User Interface (GUI)

Foresee offers a GUI powered by an internal Python engine for intuitive usage. While most of *Foresee*'s features are available through the GUI, users can also opt to interact with *Foresee* using the command line. In either mode, users must supply Python (and Verilog) code when creating a Module not present in *Foresee*.

Foresee offers three views through its GUI: *Overview*, *Pipeline*, and *Analysis*. Figure 2 shows the Overview page, which is the main view on *Foresee*'s launch. From here, users can create or import new pipelines and Modules, and launch various algorithm or hardware analyses.

Figure 3 shows the Pipeline page, which is the workspace where Modules are connected to form pipelines. Modules include inputs (synthetic data generators, or files from datasets), computational blocks, and outputs (files to write the outputs of signal processing). Modules can be added from the previously defined collection (see the example in Figure 3), or can be created anew. Selecting a module lists the options that can be configured for that module in this pipeline. Figure 3 shows the available options for a pre-defined Fast Fourier Transform (FFT) Module. Different modules can be connected with wires.

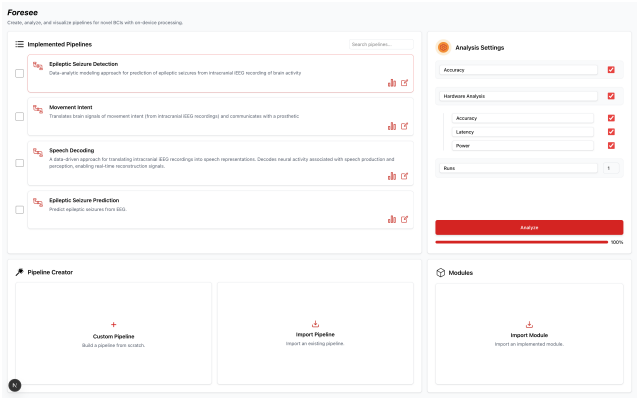


Fig. 2: *Foresee*'s Overview page (*i.e.*, its launch page).

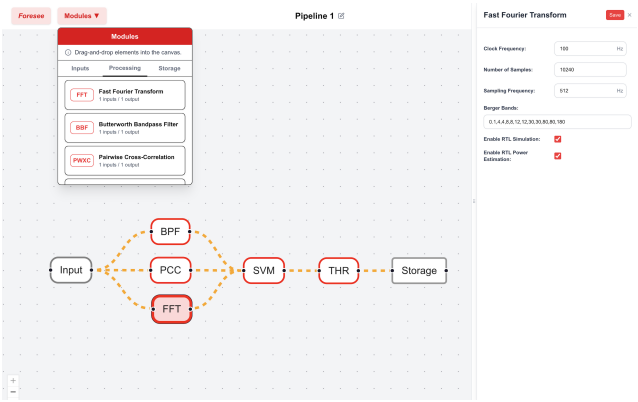


Fig. 3: *Foresee*'s Pipeline page showing an example pipeline, with the Modules window open and an FFT module selected.

Foresee also supports creating new modules. When this option is chosen from the Overview, users can select the Python and hardware files that define the new module.

Once a pipeline is specified, users can analyze Python descriptions for functional operation and accuracy, or with the hardware implementations for power and latency analyses (Section III-B). Figure 4 shows the analysis page, which presents the results of the user-specified measurements and comparisons. All data can be exported in a Comma-Separated Value (CSV) file for offline analysis.

By default, *Foresee* uses open-source tools and can integrate any semiconductor fabrication technology, silicon process node, and design tools to analyze power and latency. Our

evaluations in Section III-B assume a default process technology (130 nm), to enable tractable presentation of results.

Fig. 4: *Foresee*'s Analysis page showing sample results.

B. Forsee's Analysis Modes

Foresee currently supports three types of analyses. These are: Python execution for algorithmic accuracy, hardware latency (and accuracy), and hardware power estimation. Python execution is the simplest, and runs the modules natively on any supplied datasets to obtain accuracy values.

For hardware latency and accuracy, *Foresee* internally runs a cycle-accurate hardware simulation of each Module on the provided data using iverilog [35], which is an open source tool for Verilog simulations.

When users need power estimation, *Foresee* uses additional tools and steps because accurate power modeling depends on detailed hardware circuit and activity information. First, *Foresee* converts the Verilog description of the Module into circuits that are amenable for hardware fabrication. This is done with Yosys [36], an open-source tool which also takes as input, the fabrication process-specific information (termed the process design kit by the foundry). Then, *Foresee* uses iverilog to obtain the activation of various parts of the hardware circuits on the provided data. Next, *Foresee* uses another tool, OpenSTA [3] for timing and power analysis, based on this information. OpenSTA reads the hardware circuit, the fabrication process-specific information, and the hardware activation data to compute the total power that the Module consumes. OpenSTA is open-source and is validated against real hardware to produce highly accurate measurements.

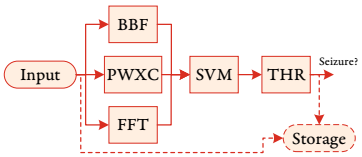
Overall, *Foresee* relies on reliable methodologies and tools to perform accurate analyses. *Foresee* also supports using custom tools instead of iverilog, Yosys, or OpenSTA, offering flexibility for users.

C. Open Hardware Library

Foresee is packaged with an open collection of hardware implementations for several signal processing modules. These are: fast Fourier transform (FFT), pairwise cross correlation (PWXC), Butterworth bandpass filter (BBF), support vector machine (SVM), thresholding block (THR), Teager-Kaiser energy operator (TKEO), and signal averaging (AVG). We envision adding more implementations to this library of hardware modules, and invite community submissions.

IV. Showcasing *Foresee* through an Example

We demonstrate *Foresee* by using it to compare two methods of seizure detection based on prior work [9, 12, 27], from the perspective of both algorithmic accuracy and on-BCI processing. The computational steps of each are shown in Figure 5, using the names of signal processing blocks from Section III-C. More details about the configurations of individual signal processing steps are available in respective references. We additionally add a storage element that captures the inputs and outputs for archival.



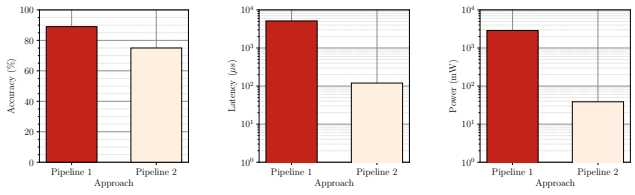
(a) Pipeline 1 based on [27].

(b) Pipeline 2 based on [9, 12].

Fig. 5: Seizure detection pipelines based on prior work to be created and compared with *Foresee*.

Following *Foresee*'s workflow in Figure 1, we begin by selecting each pipeline and describing its modules. For example, consider Pipeline 1. We navigate to the workspace by selecting the "Custom Pipeline" option on the Overview page (Figure 2). Then, from the Modules menu in the Pipeline page 3, we find that the *Foresee* hardware library has all the modules we need. We add the required modules and specify their configuration (*e.g.*, the Berger bands for the BBF module) by selecting them individually. For the SVM module, we identify its parameters by training it separately, since *Foresee* does not yet support native training, and specify those values too. We use data from [7] to train the SVM. Since we need a data source to measure accuracy, we also add a source, and specify our validation data. Next, we proceed to the Analysis page where we see the results. We repeat the process for Pipeline 2.

Foresee has the option to summarize results from multiple analyses, and Figure 6a shows the accuracy comparison we obtain. We find that Pipeline 1 has slightly higher accuracy, since it uses more features in the incoming data.



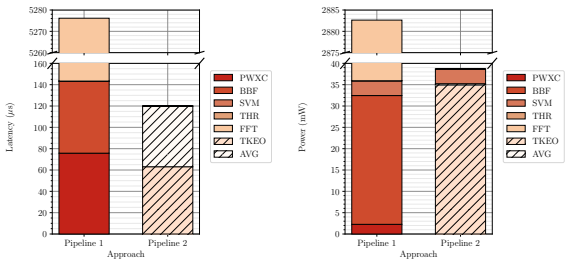
(a) Accuracy of detection. (b) Hardware latency. (c) Power consumption.

Fig. 6: Comparing two seizure detection pipelines for on-device processing with *Foresee*.

Once we are satisfied with the algorithmic accuracy of our pipelines, we then evaluate their suitability for hardware

implementation. We return to the pipelines we saved, and simply select hardware latency and power measurements in the Analysis page. *Foresee* automatically selects the hardware files for the modules and runs these analyses. Figures 6b and 6c show these results. We find that while both pipelines are quite fast (μs–ms), Pipeline 1 is about 50$\times$ slower than Pipeline 2 because it uses more complex computations. For the same reason, it also consumes nearly 3 W of power that is 100$\times$ higher than that of Pipeline 2, which consumes only 30 mW of power. Following these results, users can now easily identify the pipeline they would choose for the goals they have with the BCI. For example, for the implanted case, one could select Pipeline 2 for on-BCI processing since it is low power, while Pipeline 1 as we configured is unsuitable due to its higher power consumption.

Foresee also provides a detailed breakdown of latency and power per-module to localize inefficiency and direct co-design and optimization effort. Figure 7 shows this analysis for the two pipelines. Based on this data, we identify that the FFT module is responsible for the significantly high power consumption of Pipeline 1, followed by BBF. Indeed, we used a large 8192-point FFT with on-chip memory for our hardware that contributed to high power consumption. This insight could be used to guide the optimization in the algorithm to explore less computationally intensive alternatives.



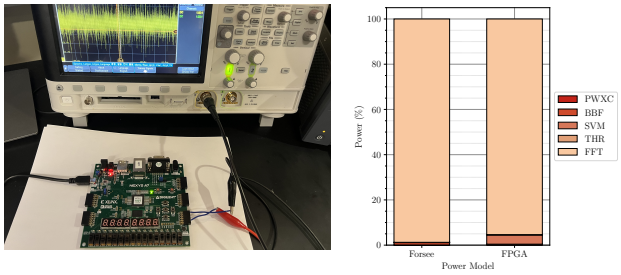
(a) Latency details. (b) Power details.

Fig. 7: Triaging on sources of inefficiency with *Foresee*.

Throughout this example workflow, we could specify different pipelines, seamlessly iterate between algorithm and hardware analysis, and obtain accurate results. The entire process could be run within the span of a few minutes. Additionally, we could quickly locate sources of inefficiency in the pipelines to focus further optimization strategies that are crucial for on-BCI processing.

A. FPGA validation

Foresee uses community-validated tools like OpenSTA for power and latency measurement. Nonetheless, to confirm *Foresee*'s ability to accurately capture hardware power consumption, we prototype the pipelines described in our example workflow on an actual FPGA test board (Xilinx Nexys A7) and validate the measured power against *Foresee*'s reported values. This setup is shown in Figure 8a. Figure 8b shows the relative breakdown of power consumption measured on the FPGA alongside those reported by *Foresee* (from Figure 7b). These results are close, with the slight difference attributed to FPGA measurement overhead, confirming *Foresee*'s accuracy.



(a) FPGA test board. (b) Power breakdown validation.

Fig. 8: Validating *Foresee* power measurements against prototyped FPGA measurements.

V. Discussion

We presented the utility of *Foresee* in advancing on-device processing for BCIs. Prior work has developed tools for other aspects of BCIs; *e.g.*, algorithmic performance and signal analysis [2, 10, 11, 25]. But, *Foresee* uniquely focuses on processor co-design and exploration. *Foresee* is open-source allowing the community to contribute and build on.

A key design choice in *Foresee* is to use a different level of design description and abstraction for algorithmic and hardware design tasks that is natural for the respective goals, such as Python for the former and Verilog for the latter, instead of forcing a common representation that is non-intuitive for either. A consequence of this choice is that the processing designs that *Foresee* can capture are limited to those that stream data through a sequence of blocks in a structured manner as described earlier. While this is restrictive for general processors, it is not so for BCIs where the computations are naturally organized as streaming dataflow pipelines that *Foresee* supports. Hence, we find this design choice in favor of intuitive design development acceptable.

We will continue to develop and support *Foresee*, and add more functionality to enhance its utility. On the algorithmic front, we are exploring methods to support plugging into frameworks like PyTorch for emerging neural network-based BCI algorithms. For hardware development, we are working to include automated checks for thermal and power compliance, and holistic hardware modeling. For example, one feature that we are currently integrating is the modeling of storage on device. On-BCI storage helps collect neural data without relying on external connections, and can also enable new BCI algorithms [31]. However, there are many parameters to consider in identifying a suitable storage medium for BCIs, such as their non-volatility (ability to hold data without power consumption), device longevity, and area. Figure 9 shows some possible non-volatile memory (NVM) storage technologies and their characteristics for Pipeline 1, obtained from NVM modeling tools like NVMExplorer [23]. The technologies modeled (described in [17]) are STT (spin-transfer torque), PCM (phase-change memory), FeFET (ferroelectric field-effect transistor), and RRAM (resistive random access memory). Due to the constraints of on-BCI processing, the choice of storage technology and size must also be optimized with processor design [33]. We plan to integrate such tools and analyses in *Foresee*.

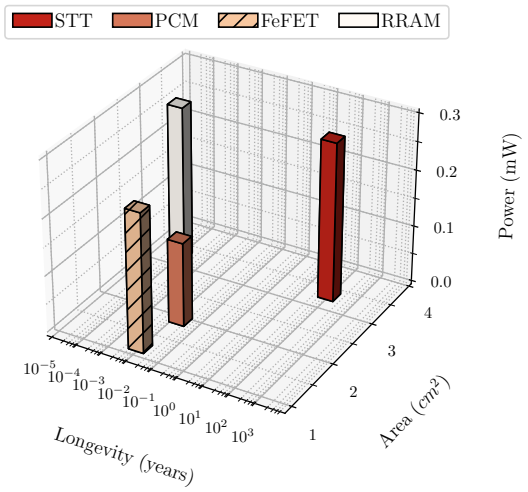


Fig. 9: NVM choices and design space for Pipeline 1.

One limitation of *Foresee* is that it supports a limited set of hardware blocks currently. However, *Foresee* is designed to naturally support a community-wide effort to add to these hardware blocks with additional Verilog description. Our eventual goal is to automate generation of Verilog modules from Python, offering even more flexibility and simplicity for users. In the interim, we plan to create an open repository for the hardware blocks for the community to submit, keeping pace with the development of new BCI algorithms, and curate these to ensure usability with *Foresee*.

Foresee also does not currently support advanced hardware design features like clock-generation and management for power, since it is aimed at early-stage hardware exploration. However, the hardware designs described in *Foresee* can easily be ported to other existing design automation software to incorporate such mechanisms. Understanding how such features can be incorporated directly into *Foresee* is an exciting design automation research task.

VI. Conclusion

This paper presented *Foresee*, a new framework that bridges the gap between computational neuroscience and computer systems in the service of on-BCI processing. Through this unified and modular platform that offers convenience, accuracy and productivity, *Foresee* could serve the needs of the BCI community in developing new-generation BCIs.

References

[1] Advanced Micro Devices, Inc. *PYNQ*. 2024. URL: https://www.pynq.io/.

[2] Agency Enterprise. *Neural Data Simulator*. 2023. URL: https://github.com/agencyenterprise/neural-data-simulator.

[3] Tutu Ajayi et al. "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain". In: *Proceedings of Government Microcircuit Applications and Critical Technology Conference*. 2019.

[4] ARM Limited. *Cortex-M23: Smallest, Lowest-Power MCU with TrustZone Security*. Product Documentation. ARM, 2025. URL: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m23.

[5] Shunning Batten, Jiang Christopher, and Torng Christopher. "An Open-source Python-based Hardware Generation, Simulation, and Verification Framework". In: *Workshop on Open-Source EDA Technology*. 2018.

[6] Nathan Binkert et al. "The Gem5 Simulator". In: *SIGARCH Comput. Archit. News* 39.2 (Aug. 2011). DOI: 10.1145/2024716.2024718.

[7] Alessio Burrello et al. "Laelaps: An Energy-Efficient Seizure Detection Algorithm from Long-term Human iEEG Recordings without False Alarms". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019. DOI: 10.23919/DATE.2019.8715186.

[8] Adam Conway. *Qualcomm Snapdragon 8 Gen 3 review*. URL: https://www.xda-developers.com/qualcomm-snapdragon-8-gen-3-review/#power-efficiency.

[9] Mark J Cook et al. "Prediction of Seizure Likelihood with a Long-term, Implanted Seizure Advisory System in Patients with Drug-resistant Epilepsy: a First-in-man Study". In: *The Lancet Neurology* 12.6 (2013), pp. 563–571. DOI: 10.1016/S1474-4422(13)70075-9.

[10] J. T. Costello. *BCISimulator: A Simple Closed-Loop BCI Simulator for Testing Real-Time Neural Decoding Algorithms*. 2023. URL: https://github.com/jtcostello/bcisimulator.

[11] Anne-Sophie Dubarry et al. "An Open-source Toolbox for Multi-patient Intracranial EEG Analysis (MIA)". In: *NeuroImage* 257 (2022), p. 119251. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2022.119251.

[12] Nicolas Gaspard et al. "Automatic Detection of Prominent Interictal Spikes in Intracranial EEG: Validation of an Algorithm and Relationsip to the Seizure Onset Zone". In: *Clinical Neurophysiology* 125.6 (2014), pp. 1095–1103. DOI: 10.1016/j.clinph.2013.10.021.

[13] Reid R. Harrison et al. "Wireless Neural Signal Acquisition with Single Low-power Integrated Circuit". In: *IEEE International Symposium on Circuits and Systems*. May 2008. DOI: 10.1109/iscas.2008.4541776.

[14] Intel Corporation. *Intel® Simics® Simulator*. 2025. URL: https://www.intel.com/content/www/us/en/developer/articles/tool/simics-simulator.html.

[15] Shunning Jiang, Berkin Ilbeyi, and Christopher Batten. "Mamba: Closing the Performance Gap in Productive Hardware Development Frameworks". In: *Annual Design Automation Conference*. 2018.

[16] Ioannis Karageorgos et al. "Hardware-software Co-design for Brain-computer Interfaces". In: *International Symposium on Computer Architecture (ISCA)*. 2020. DOI: 10.1109/ISCA45697.2020.00041.

[17] Saeed Kargar and Faisal Nawab. "Challenges and future directions for energy, latency, and lifetime improvements in NVMs". In: *Distributed and Parallel Databases* 41.3 (2023), pp. 163–189.

[18] Mikhail A. Lebedev and Miguel A. L. Nicolelis. "Brain-Machine Interfaces: From Basic Science to Neuroprostheses and Neurorehabilitation". In: *Physiological Reviews* 97.2 (Apr. 2017), pp. 767–837. DOI: 10.1152/physrev.00027.2016.

[19] P.S. Magnusson et al. "Simics: A Full System Simulation Platform". In: *Computer* 35.2 (2002), pp. 50–58. DOI: 10.1109/2.982916.

[20] Elon Musk and Neuralink. "An Integrated Brain-machine Interface Platform with Thousands of Channels". en. In: *J. Med. Internet Res.* 21.10 (Oct. 2019), e16194.

[21] NVIDIA Corporation. *Get Started With Jetson Nano Developer Kit*. Developer Documentation. NVIDIA. 2025. URL: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.

[22] Gerard O'Leary et al. "NURIP: Neural Interface Processor for Brain-State Classification and Programmable-Waveform Neurostimulation". In: *IEEE Journal of Solid-State Circuits* 53.11 (2018), pp. 3150–3162. DOI: 10.1109/JSSC.2018.2869579.

[23] Lillian Pentecost et al. "NVMExplorer: A Framework for Cross-Stack Comparisons of Embedded Non-Volatile Memories". In: *International Symposium on High-Performance Computer Architecture (HPCA)*. 2022. DOI: 10.1109/HPCA53966.2022.00073.

[24] Adrian Sampson, James Bornholt, and Luis Ceze. "Hardware-Software Co-Design: Not Just a Cliché". In: *1st Summit on Advances in Programming Languages (SNAPL 2015)*. Vol. 32. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 262–273. DOI: 10.4230/LIPIcs.SNAPL.2015.262.

[25] Science Corporation. *Nexus: Neural Engineering Research Platform*. 2025. URL: https://science.xyz/technologies/nexus/.

[26] Claudia Serrano-Amenos et al. "Thermal Analysis of a Skull Implant in Brain-Computer Interfaces". In: *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, July 2020. DOI: 10.1109/embc44109.2020.9175483.

[27] Han-Tai Shiao et al. "SVM-Based System for Prediction of Epileptic Seizures From iEEG Signal". In: *IEEE Transactions on Biomedical Engineering* 64.5 (May 2017), pp. 1011–1022. ISSN: 1558-2531. DOI: 10.1109/tbme.2016.2586475.

[28] Uisub Shin et al. "NeuralTree: A 256-Channel 0.227-μJ/Class Versatile Neural Activity Classification and Closed-Loop Neuromodulation SoC". In: *IEEE Journal of Solid-State Circuits* 57.11 (2022), pp. 3243–3257. DOI: 10.1109/JSSC.2022.3204508.

[29] John D Simeral et al. "Home Use of a Percutaneous Wireless Intracortical Brain-Computer Interface by Individuals With Tetraplegia". In: *IEEE Transactions on Biomedical Engineering* 68.7 (2021), pp. 2313–2325. DOI: 10.1109/TBME.2021.3069119.

[30] Karthik Sriram et al. "HALO: A Hardware–Software Co-Designed Processor for Brain–Computer Interfaces". In: *IEEE Micro* 43.3 (May 2023), pp. 64–72. DOI: 10.1109/mm.2023.3258907.

[31] Karthik Sriram et al. "SCALO: An Accelerator-Rich Distributed System for Scalable Brain-Computer Interfacing". In: *International Symposium on Computer Architecture*. June 2023. DOI: 10.1145/3579371.3589107.

[32] Felice T Sun and Martha J Morrell. "The RNS System: Responsive Cortical Stimulation for the Treatment of Refractory Partial Epilepsy". In: *Expert Review of Medical Devices* 11.6 (Aug. 2014), pp. 563–572. DOI: 10.1586/17434440.2014.947274.

[33] Muhammed Ugur, Raghavendra Pradyumna Pothukuchi, and Abhishek Bhattacharjee. "Swapping-Centric Neural Recording Systems". In: *The 15th Annual Non-Volatile Memories Workshop*. 2024.

[34] Francis R Willett et al. "A high-performance speech neuroprosthesis". In: *Nature* 620.7976 (2023), pp. 1031–1036. DOI: 10.1038/s41586-023-06377-x.

[35] Stephen Williams and Michael Baxter. "Icarus Verilog: Open-source Verilog More Than a Year Later". In: *Linux J.* 2002.99 (July 2002), p. 3. ISSN: 1075-3583.

[36] Clifford Wolf, Johann Glaser, and Johannes Kepler. "Yosys-A Free Verilog Synthesis Suite". In: 2013. URL: https://api.semanticscholar.org/CorpusID:202611483.

[37] Patrick D. Wolf. "Thermal Considerations for the Design of an Implanted Cortical Brain-Machine Interface (BMI)". In: *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment* (2008). URL: https://www.ncbi.nlm.nih.gov/books/NBK3932/.

[38] Do-Yeon Yoon et al. "A 1024-Channel Simultaneous Recording Neural SoC with Stimulation and Real-Time Spike Detection". In: *2021 Symposium on VLSI Circuits*. 2021, pp. 1–2. DOI: 10.23919/VLSICircuits52068.2021.9492480.

[39] Michael J Young, David J Lin, and Leigh R Hochberg. "Brain–Computer Interfaces in Neurorecovery and Neurorehabilitation". In: *Seminars in neurology*. Vol. 41. 02. Thieme Medical Publishers, Inc. 2021, pp. 206–216.