

École polytechnique de Louvain

Hardware Accelerator Design and Algorithmic Optimization for an Ultra-Low-Power Vagus Nerve-Based Seizure Detection System

Author: **Julien DAVID**

Supervisor: **Prof. David BOL**

Readers: **Prof. Martin ANDRAUD, Sylvain FAVRESSE**

Academic year 2025–2026

Master [120] in Electrical Engineering

Abstract

Treating drug-resistant epilepsy using implantable neurostimulation devices requires reliable continuous monitoring solutions that are compatible with stringent power constraints. Vagus nerve electroneurography (VENG) has recently emerged as a promising signal source for seizure detection. However, existing approaches remain limited by their power consumption. This work aims to design an ultra-low-power seizure detection system by improving and extending a previously proposed VENG-based solution.

To this end, algorithmic/software optimizations and hardware acceleration are jointly explored. The original normalized cross-correlation (NCC)-based template matching algorithm is adapted to better support hardware implementation, while software optimizations are introduced to reduce power consumption. In addition, several hardware architectures are investigated, leading to the design of a parallel-template NCC hardware accelerator.

The proposed approach achieves a power consumption of 2.45 mW for a single-template software-only implementation of the seizure detection algorithm. Compared to prior work, this corresponds to a $\times 4$ reduction in execution time and a $\times 3.77$ reduction in power consumption. Furthermore, the introduced optimizations enable real-time processing of up to 7 templates instead of one. The hardware accelerator exhibits an estimated power consumption of 53.48 μ W at 1 MHz while maintaining high computational accuracy. System-level projections on the BREWER platform indicate a total power consumption of 502.37 μ W.

These results demonstrate that efficient seizure detection under strict ultra-low-power constraints can be achieved through a co-design approach combining software optimizations and hardware acceleration, thereby paving the way toward practical implantable VENG-based monitoring systems.

Acknowledgments

I would first like to express my sincere gratitude to Prof. David Bol for giving me the opportunity to work on such a fascinating topic. I am especially thankful for his guidance, insightful feedback, availability, and expertise throughout this work. I am also deeply grateful for the opportunities he has offered me over the past year, and I am truly looking forward to continuing this collaboration as a PhD student.

I would like to warmly thank Sylvain Favresse for his close supervision, his helpfulness in understanding the subject, and the clarity and precision with which he answered my questions. His constant availability and guidance were highly valuable throughout this project.

I am also deeply grateful to Pol Maistriaux for sharing his expertise in digital circuit design. His support with the simulation and synthesis toolchain, the development of specialized library macros, and access to implementation platforms and laboratory instruments was essential to this work. I greatly appreciate the time he dedicated to answering my many questions. Finally, I would like to congratulate him on becoming a father.

I would like to thank my girlfriend for her continuous moral support, her genuine interest in my work, and, more broadly, for her unwavering support over the past five years.

Finally, I wish to express my deepest gratitude to my parents for encouraging me to pursue engineering studies, for their advice, and for their support, including their careful reviewing of this manuscript.

Use of AI

Artificial intelligence tools were used throughout the development of this thesis as support tools for both manuscript preparation and code development.

In the context of manuscript writing, Microsoft Copilot and LanguageTool were used to help structure certain sections, rephrase text for clarity, and improve overall readability. They were also occasionally used to identify relevant scientific literature. All generated or suggested content was systematically reviewed, validated, and adapted to ensure accuracy, coherence, and alignment with the scientific objectives of the work.

For code development, GitHub Copilot was used to facilitate code writing through autocompletion. AI was also consulted to assist in generating SystemVerilog testbenches during the development process, as well as to provide guidance on SystemVerilog syntax and support debugging efforts. All generated code was carefully reviewed, tested, and validated before integration into the project.

In all cases, AI tools were used as supportive aids, and the author retained full responsibility for the design choices, implementation, results, and conclusions presented in this thesis.

Contents

Introduction	VII
1 Previous work in VENG-based seizure detection	1
1.1 VENG-based seizure detection algorithm	1
1.1.1 Action potential detection	2
1.1.2 Seizure detection	3
1.2 Previous low-power implementation	5
1.2.1 Processing pipeline	5
1.2.2 Timing and power results	7
1.2.3 Identified bottlenecks	8
1.3 Improvements targeted in this work	8
1.3.1 Performance targets	8
1.3.2 NCC hardware acceleration	9
1.3.3 Algorithm validation	10
2 Algorithm and software optimization	13
2.1 Algorithmic optimization	13
2.1.1 Removal of the clustering step	14
2.1.2 Template normalization	15
2.1.3 Correlation-based proxy for AP amplitude	15
2.1.4 Final optimized pipeline	17
2.2 Low-power implementation	18
2.2.1 Integration with the targeted platform	19
2.2.2 Software implementations and real-time scheduling	19
2.2.3 Word-length reduction	22
2.3 Implications for the hardware accelerator	23

3	Design and implementation of a normalized cross-correlation hardware accelerator	25
3.1	Design assumptions and accelerator interface	25
3.2	Architecture exploration and selection	27
3.2.1	Candidate architectures and dataflow	27
3.2.2	First-order cost model	37
3.2.3	Architecture comparison and selected design	41
3.3	Microarchitecture of the selected parallel-templates design	42
3.3.1	Architecture overview	42
3.3.2	Input buffering and local memory	43
3.3.3	Control datapath	45
3.3.4	Correlation datapath	47
3.3.5	Signal energy datapath	47
3.3.6	RSQRT datapath	48
3.3.7	NCC formation datapath	49
3.3.8	Implementation summary	50
3.4	Verification approach and implementation toolflow	50
4	Characterization and discussion	53
4.1	Software implementation characterization	53
4.1.1	Evaluation and measurement setup	53
4.1.2	Functional evaluation	54
4.1.3	Execution time characterization and scalability	55
4.1.4	Power characterization	59
4.1.5	Memory utilization	61
4.2	Hardware accelerator	62
4.2.1	Evaluation and synthesis setup	62
4.2.2	Numerical validation	63
4.2.3	Post-synthesis power and area	64
4.2.4	Timing closure, latency and throughput	65
4.3	Integration perspective and system comparison	66
4.3.1	Projection on the BREWER platform	66
4.3.2	Comparison with neural spike processing systems	68
	Conclusion	68

Acronyms

ADC analog-to-digital converter

AFE analog front-end

AP action potential

DBE digital back-end

DMA direct memory access

DRE drug-resistant epilepsy

DSP digital signal processing

EEG electroencephalography

LMEM local memory

LUT lookup table

MAC multiply and accumulate

MCU microcontroller unit

MSB most significant bit

NCC normalized cross-correlation

PPA power, performance, and area

PTZ pentylenetetrazol

RSQRT reciprocal square root

RTL register-transfer level

SIMD single instruction, multiple data

SMU source measure unit

SoC system on a chip

SRAM static random access memory

TDL tapped delay line

ULP ultra-low-power

VENG vagus nerve electroneurogram

VNS vagus nerve stimulation

Introduction

Neurostimulation technologies have advanced considerably over the past two decades. Research activity in this field has grown steadily, and the clinical role of neurostimulation devices is progressively evolving from a last-resort solution toward an option considered earlier in disease treatment. However, designing such devices remains highly challenging. In battery-powered implantable neurostimulators intended for chronic diseases, constraints on size, battery capacity, power consumption, and long-term reliability are central design considerations [1].

Among chronic diseases that may benefit from implantable neurostimulators, epilepsy is a particularly relevant case because of its high global burden and the large portion of patients for whom antiseizure medicines are not effective [2]. About one-third of patients with epilepsy develop drug-resistant epilepsy (DRE). According to the International League Against Epilepsy, DRE is defined as the failure of adequate trials of two tolerated and appropriately chosen antiseizure medication schedules to achieve sustained seizure freedom [3]. For these patients, persistent seizures are not only a therapeutic problem but also a major source of disability, comorbidity, and premature mortality.

Vagus nerve stimulation (VNS) is an established neuromodulation therapy for drug-resistant epilepsy [4]. The vagus nerve is particularly relevant in this context because it plays a central role in autonomic regulation and conveys bidirectional information between the body and the brain. As seizures are often associated with autonomic manifestations, their occurrence may induce detectable modulations in vagal neural activity. The electrical activity recorded from the vagus nerve, referred to as the vagus nerve electroneurogram (VENG), has therefore emerged as a promising candidate biomarker for seizure detection and closed-loop VNS [5].

However, the implementation of VENG-based seizure detection into an implantable device remains limited by computational and power constraints. In [6], a first complete VENG-based seizure detection system was implemented on a microcontroller unit (MCU). The proposed pipeline combines action potential (AP)

detection through the template-matching method of [7] with a seizure detection metric based on AP amplitude and rate, as introduced in [5]. Results obtained on an Apollo3 Blue system-on-chip (SoC) with an ARM Cortex-M4 processor indicate that both algorithmic and implementation-level improvements are necessary to reduce power consumption and to meet the real-time requirements of continuous VENG monitoring. In particular, to fulfill the real-time constraint, the AP detection stage is intentionally limited to a single template in the implemented system, although up to three templates were originally supported. This configuration reflects the memory- and timing-constrained nature of the embedded platform but also suggests that enabling the use of a larger number of templates could be beneficial for seizure detection quality. Allowing for such an extension under the same constraints therefore constitutes an important objective of this work. Moreover, further developments of the overall detection algorithm are needed to improve the reliability of the system. These limitations motivate the need for both algorithmic refinements and more specialized hardware architectures for implantable VENG-based seizure detection.

In this context, the objective of this work is to improve the practical feasibility of VENG-based seizure detection for implantable applications by reducing its computational cost and enabling more energy-efficient implementations. To this end, the work is structured around two complementary contributions. First, the seizure detection algorithm and its embedded implementation are optimized in order to improve detection performance while reducing power consumption and verifying real-time feasibility. Second, a dedicated hardware accelerator is designed for the normalized cross-correlation (NCC) operation used in the AP detection stage, with the aim of further reducing the computational burden and energy cost of the system.

On the algorithmic and embedded implementation side, this work revisits the existing detection pipeline and proposes optimizations at both the signal-processing and deployment levels. These optimizations include fixed-point word-length reduction, the use of hardware filters specific to the BREWER chip introduced in [8], and several modifications to the AP detection algorithm. The assessment of the detection capabilities relies on a dataset provided by the Université Libre de Bruxelles, consisting of VENG recordings of induced seizures in rats together with annotations of the animal state derived from EEG recordings. The resulting developments are evaluated on a STM32 Nucleo board from STMicroelectronics, on which the full detection chain is implemented and characterized. In particular, power consumption measurements are carried out and the real-time feasibility of continuous VENG monitoring is assessed. This part of the work aims to quantify the extent to which algorithmic and implementation-level refinements can improve

the suitability of the system for chronic ultra-low-power (ULP) operation.

On the hardware side, this thesis focuses on the NCC operation, which constitutes the computational core of the template-matching stage used for AP detection. A dedicated hardware accelerator is therefore proposed to offload this operation from the embedded processor. To identify the most suitable solution, four architectures are explored and compared according to their expected implementation cost and performance. One of them is then selected and implemented from register-transfer level (RTL) description through simulation and logic synthesis using Siemens QuestaSim and Synopsys Design Compiler, leading to post-synthesis power, performance, and area (PPA) results.

The selected design occupies a cell area of 0.062 mm^2 and achieves a power consumption of $53.48 \text{ }\mu\text{W}$ at a clock frequency of 1 MHz when processing eight templates, while the optimized hardware accelerator-compatible embedded implementation consumes 1.65 mW . Furthermore, the power consumption of the fully software-based optimized implementation is reduced by a factor of 3.77 compared to [6]. Taken together, these results highlight the trade-offs between algorithmic efficiency, software deployment, and hardware acceleration for ULP VENG-based seizure detection. Overall, the proposed optimizations improve the efficiency of the detection chain and bring its implementation closer to the requirements of an implantable ULP seizure detection system. More generally, this work contributes to bridging the gap between proof-of-concept VENG-based monitoring and its integration into practical closed-loop neurostimulation devices.

The remainder of this manuscript is organized as follows. Chapter 1 presents the initial VENG-based seizure detection algorithm and its first embedded implementation, providing the baseline on which this work builds. Chapter 2 introduces the proposed optimizations of both the detection algorithm and its embedded realization, with a focus on improving efficiency, reducing power consumption, and verifying real-time feasibility. Chapter 3 describes the design and implementation of the dedicated hardware accelerator for the NCC operation used in the AP detection stage, including the exploration and comparison of several architectural solutions. Finally, Chapter 4 reports the validation and characterization of the proposed developments, bringing together the results obtained for both the optimized embedded system and the hardware accelerator.

Chapter 1

Previous work in VENG-based seizure detection

This opening chapter reviews prior works on which the present work builds. It first describes the baseline VENG-based seizure detection algorithm. It then discusses the continuous-acquisition embedded implementation reported in [6] and identifies its main computational and system-level bottlenecks. Finally, the performance targets adopted in this work are formulated together with a metric used to quantify algorithmic improvements, and the principal contribution of this work, namely the hardware acceleration of the normalized cross-correlation operation, is motivated.

1.1 VENG-based seizure detection algorithm

The baseline algorithm aims to detect and, when feasible, anticipate seizure onsets from VENG recordings. It operates as a two-stage pipeline, summarized in Figure 1.1. In the first stage (AP detection), the VENG signal is band-pass filtered and a sparse event stream is extracted by identifying candidate APs and estimating event features, notably timestamps and amplitudes. The second stage (seizure detection) aggregates the event stream over time to compute a detection metric from which a threshold-based classifier derives a binary seizure decision. The following subsections describe the AP detection and seizure detection stages in detail.

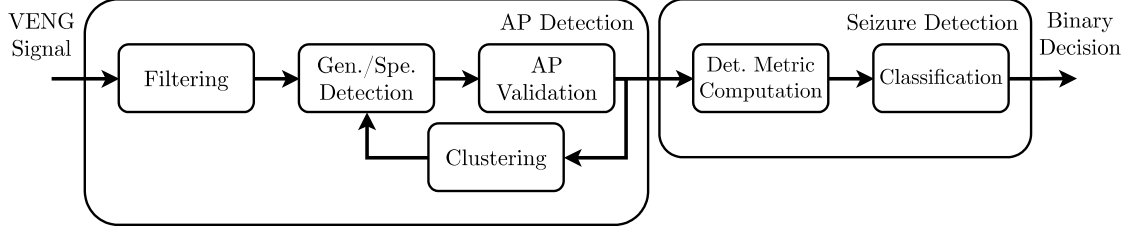


Figure 1.1: High-level pipeline of the baseline algorithm. The VENG signal is filtered and processed by the AP detection stage to produce an event stream, which is then aggregated to compute the detection metric m and a binary seizure decision. Gen., Spe., and Det. denote generic, specific, and detection, respectively.

1.1.1 Action potential detection

The AP detection approach follows [7] and is based on template matching using the normalized cross-correlation (NCC). NCC provides a similarity score between an AP template and local segments of the VENG signal as the template slides over time. Equation (1.1) defines the NCC at lag τ as the normalized inner product between p_i , an AP template of length M , and the length- M segment of the VENG signal s starting at τ , divided by the product of their Euclidean norms.

$$\text{NCC}_i[\tau] = \frac{\sum_{n=0}^{M-1} s[n + \tau] p_i[n]}{\sqrt{\sum_{n=0}^{M-1} s^2[n + \tau]} \sqrt{\sum_{n=0}^{M-1} p_i^2[n]}} \quad (1.1)$$

where τ is chosen such that $s[n + \tau]$ is defined for $n = 0, \dots, M - 1$.

AP detection consists of the following steps:

Filtering The signal is band-pass filtered using a second-order Butterworth filter (300 Hz–3 kHz) applied in forward–backward mode to achieve zero-phase distortion.

Generic Detection From the filtered signal, an NCC trace is computed using a generic template, p_{gen} , representing a typical vagus nerve AP shape [9]. Because the NCC is normalized, it is invariant to amplitude scaling. Additionally, the absolute value is taken to detect both positive and inverted AP waveforms. As such, this yields similarity values in $[0, 1]$, where 1 indicates a perfect match. Candidate APs are detected as local maxima of $|\text{NCC}_{\text{gen}}[\tau]|$ exceeding a fixed threshold of 0.8. To avoid multiple detections of the same event and overlapping matches, successive

candidates must be separated by at least $M + 0.1 \text{ ms} \cdot f_s$ samples, where f_s is the sampling frequency. The resulting peak locations define the candidate AP timestamps in the filtered signal.

AP Validation For each candidate AP, the amplitude A_{AP} is measured as the difference between the maximum and the minimum value within a window of M samples centered on the candidate timestamp. The local RMS amplitude RMS_{loc} of the filtered signal is computed over a 500 ms window. Two adaptive amplitude thresholds are then defined as $A_{\min} = k_{\min} \text{RMS}_{\text{loc}}$ and $A_{\max} = k_{\max} \text{RMS}_{\text{loc}}$, where $k_{\min}$ and $k_{\max}$ are parameters of the algorithm. Candidates with $A_{AP} < A_{\min}$ are rejected as noise, while candidates with $A_{AP} > A_{\max}$ are rejected as artifacts. This leaves only validated AP events.

Clustering Next, validated APs are clustered in the time domain using K-means. The aim is to group APs with similar shapes and to identify waveform patterns that recur in the signal. The number of clusters is selected automatically to adapt to the number of distinct AP morphologies. The resulting centroid waveforms are used as specific templates for the next step.

Specific Detection Similarly to the generic detection, a specific detection step is performed using the template set $\{p_i\}$ obtained from the clustering step. For each template p_i , a NCC trace $\text{NCC}_i[\tau]$ is computed. Candidate APs are detected as local maxima of $|\text{NCC}_i[\tau]|$ using the same detection threshold and minimum-separation criterion as in the generic step. Finally, the AP validation step described above is applied to the resulting candidates.

1.1.2 Seizure detection

The seizure detection technique proposed in [5] builds on an early version of the AP detection method described above. Starting from the validated AP event stream, AP firing rate and amplitude are aggregated over time to compute the detection metric m . Seizure activity is then inferred using a threshold-based binary decision rule applied to this metric. A cross-validation methodology is additionally introduced to tune key algorithm parameters.

The seizure detection stage consists of the following steps:

Detection Metric Computation Validated APs are aggregated using two adjacent sliding windows: a background window (bg) of duration T_{bg} and a foreground window (fg) of duration T_{fg} . At each analysis time, the foreground window corresponds to the most recent T_{fg} seconds, while the background window corresponds to the immediately preceding T_{bg} seconds. Following [5], the window durations are set to $T_{bg} = 90$ s and $T_{fg} = 10$ s.

For each window, two quantities are computed: the mean absolute AP amplitude

$$U_{bg} = \frac{1}{N_{bg}} \sum_{i \in \mathcal{W}_{bg}} |A_{AP,i}|, \quad U_{fg} = \frac{1}{N_{fg}} \sum_{i \in \mathcal{W}_{fg}} |A_{AP,i}|, \quad (1.2)$$

and the AP firing rate

$$f_{bg} = \frac{N_{bg}}{T_{bg}}, \quad f_{fg} = \frac{N_{fg}}{T_{fg}}, \quad (1.3)$$

where $\mathcal{W}_{bg}$ and $\mathcal{W}_{fg}$ denote the sets of AP indices falling within the background and foreground windows, and $N_{bg} = |\mathcal{W}_{bg}|$ and $N_{fg} = |\mathcal{W}_{fg}|$ are the corresponding event counts.

From these quantities, two relative changes are defined:

$$U_r = \frac{U_{fg}}{U_{bg}}, \quad f_r = \frac{f_{fg}}{f_{bg}}. \quad (1.4)$$

The relative changes are normalized by their standard deviations, σ_U and σ_f , computed over the interval [100, 200] s from the start of each recording to ensure both bg and fg windows are fully defined:

$$\tilde{U}_r = \frac{U_r - 1}{\sigma_U}, \quad \tilde{f}_r = \frac{f_r - 1}{\sigma_f}. \quad (1.5)$$

Finally, the normalized quantities are combined into the detection metric

$$m = (\tilde{U}_r + 1) (\tilde{f}_r + 1). \quad (1.6)$$

Outside seizure activity, m is expected to remain approximately constant and close to 1. Meanwhile, increases in AP amplitude and/or firing rate lead to larger values of m . An example of a detection metric is shown in Figure 1.3.

Classification Based on the detection metric, a simple threshold-based classifier is defined. The seizure threshold θ_{seizure} is computed from the mean and standard deviation of the metric, μ_m and σ_m , estimated over a certain interval:

$$\theta_{\text{seizure}} = \mu_m + k\sigma_m, \quad (1.7)$$

where k is a tunable standard-deviation multiplier. A seizure is reported when $m[n]$ exceeds θ_{seizure} for at least n_τ consecutive samples, where n_τ is an additional tunable parameter.

1.2 Previous low-power implementation

In [6], the baseline algorithm is implemented on a low-power embedded platform using continuous acquisition, which necessitates several modifications from offline, file-based processing. These adaptations are motivated by limited compute and memory resources, timing requirements imposed by buffered acquisition, and the objective of minimizing average power consumption. The following subsections describe the resulting embedded processing pipeline, summarize the measured timing and power results on the target platform, and identify the dominant bottlenecks that motivate the improvements targeted in this work.

1.2.1 Processing pipeline

To satisfy real-time constraints in a continuous acquisition context, various modifications are introduced compared to offline processing. In [6], data are processed in successive analysis windows defined by the acquisition buffers. Each window contains 500 ms of signal, corresponding to 10 000 samples at 20 kS/s. The AP detection pipeline is then executed on each analysis window.

Within each 500 ms analysis window, the acquired signal is first band-pass filtered and then processed by template matching to extract candidate AP events. Unlike the offline formulation in [7], the embedded implementation does not maintain a permanent two-stage “generic then specific” detection flow. Instead, template matching operates solely on the currently active template set. During an initialization phase, when no recording-specific templates are yet available, the generic template is used to bootstrap the template-based detection. Thereafter, the template set is periodically refreshed using clustering performed on a rolling collection of recently validated AP waveforms. Following template matching, the AP validation stage is applied. The event stream is then forwarded to the seizure detection stage, where the detection metric computation and the threshold-based classification remain identical to the baseline method. The resulting embedded processing pipeline and its periodic template update mechanism are summarized in Figure 1.2.

The following modifications are introduced to enable continuous, real-time execution on the embedded target and are presented in the order in which they occur in the processing pipeline.

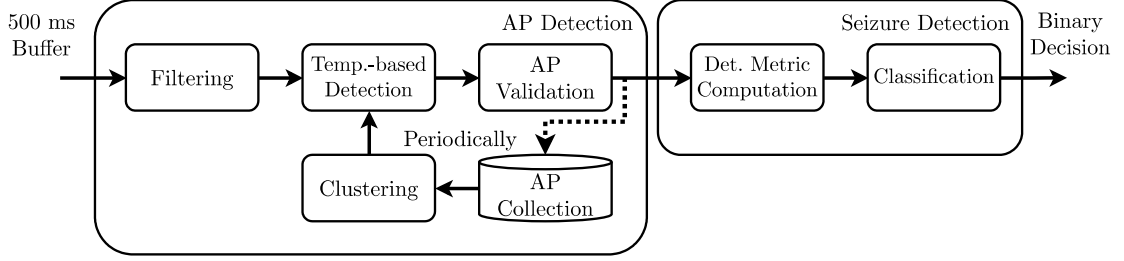


Figure 1.2: Flowchart of the embedded processing pipeline. VENG data are acquired in successive 500 ms buffers and processed through filtering, template-based AP detection, and AP validation, followed by detection metric computation and threshold-based classification to produce a binary seizure decision. In parallel, a subset of validated AP waveforms is collected over a limited time span (dotted arrow), corresponding to the clustering duration. These waveforms are clustered periodically to refresh the active template set.

- **Causal filtering:** Band-pass filtering is implemented using two cascaded second-order IIR biquad sections rather than a forward-backward zero-phase filter, which is incompatible with causal streaming execution.
- **On-the-fly template aggregation:** For multi-template detection, the individual NCC traces are not stored simultaneously. Instead, an aggregated similarity trace is formed as

$$c[\tau] = \max_i |\text{NCC}_i[\tau]|, \quad (1.8)$$

and candidate APs are detected as local maxima of $c[\tau]$. This reduces the memory footprint by avoiding one stored NCC trace per template.

- **Template norm normalization:** To reduce per-sample fixed-point operations, all templates are scaled to share a common norm. The detection threshold is scaled accordingly, allowing the per-template normalization factor to be absorbed into a single precomputed threshold term and eliminating repeated multiplications by template norms during NCC evaluation.
- **Bounded clustering resources:** Upper bounds are imposed on the number of clusters and on the number of stored AP waveforms used for clustering to fit within the available on-chip memory.
- **Rolling-sum updates:** Rolling sums are used where possible to avoid recomputation over sliding windows, notably for the signal-energy term required for NCC normalization and for the background/foreground aggregates (U_{bg} , U_{fg} , f_{bg} , and f_{fg}) in the detection metric computation.

No. of Templates	Execution time (ms)				
	Filtering	AP Detection	Clustering	Metric Comp.	Total
1	47	289	27	0.045	363
2	47	567	25	0.044	639

Table 1.1: Execution time breakdown per buffer for one- and two-template configurations. Bold entries exceed the 500 ms hard real-time constraint. AP detection includes template matching and AP validation. No execution-time measurements are reported for the final classification step. Comp. denotes computation.

In [6], the prototype implementation targets an Apollo3 Blue SoC (Ambiq Micro) featuring an ARM Cortex-M4 32-bit microcontroller core. The core clock is set to 48 MHz. To emulate a fully integrated system, the analog-to-digital converter (ADC) is configured for 14-bit resolution without oversampling. The ADC is timer-triggered at 20 kHz. The direct memory access (DMA) engine is employed using a double-buffering scheme to enable continuous data acquisition. During operation, the system alternates between processing one buffer and placing the Cortex-M4 core into deep sleep between processing bursts. To meet the computational constraints of the microcontroller, arithmetic is performed using a 32-bit fixed-point representation rather than the double-precision floating-point format used in the baseline algorithm. The implementation relies on the CMSIS-DSP library [10] for optimized digital signal processing (DSP) primitives on ARM Cortex-M processors, notably for filtering and vector operations.

1.2.2 Timing and power results

To support continuous acquisition, each 500 ms signal buffer must be processed within 500 ms, which defines a hard real-time constraint. With one template, the measured total processing time is 363 ms per 500 ms buffer, corresponding to a duty cycle of approximately 73%. Table 1.1 reports the per-buffer execution-time breakdown measured in [6].

Reported power measurements indicate an average consumption of approximately 2.15 mW for the embedded prototype in continuous operation, including acquisition through the ADC with DMA. Within this budget, the ADC and DMA account for about 0.96 mW, while the remaining consumption is attributed to the MCU core and associated digital activity. The processing energy is largely driven by the AP detection stage, which concentrates most computations and determines the active duty cycle.

1.2.3 Identified bottlenecks

Results highlight two primary bottlenecks that limit the suitability of the baseline embedded implementation for an always-on, ultra-low-power system. First, the reported average consumption of 2.15 mW remains high for long-term autonomous operation, even though direct comparisons across applications and technology platforms must be made cautiously. Nevertheless, substantially lower average power levels have been demonstrated in other always-on biomedical systems (e.g., 13.1 μ W at 1.8 V for an ECG classification SoC) [11], and ultra-low-power VENG analog front-ends (AFE) have been reported at the μ W level (e.g., 11.8 μ W at 1.2 V for a dual-channel VENG AFE) [8]. These references suggest that the milliwatt-level budget observed in [6] leaves significant room for improvement on both the analog acquisition side and the digital processing side.

Second, real-time execution constitutes a limiting constraint when extending template matching beyond a single template. As shown in Table 1.1, the hard 500 ms per-buffer deadline is met with one template but is violated with two templates. The increase is dominated by the AP detection stage, indicating that template matching and associated per-sample operations form the critical runtime bottleneck. This limitation restricts the practical use of multiple templates, which would otherwise be desirable to capture variability in AP morphologies and improve detection robustness.

1.3 Improvements targeted in this work

This section summarizes the objectives and main design choices investigated in the present work in response to the limitations observed in the prior embedded implementation. First, the performance targets are stated in terms of power, real-time capability under multi-template operation, and preservation of seizure-related information. Next, the central proposed approach, namely hardware acceleration of the NCC computations used for template matching, is introduced. Finally, the validation methodology used to quantify algorithmic impact is described, including the dataset and the metric-based evaluation protocol.

1.3.1 Performance targets

The developments proposed in this work are driven by three complementary performance targets, defined with respect to the embedded implementation reported

in [6]. A first objective is to reduce the average power consumption required for always-on operation. In the baseline implementation, power is reported in the milliwatt range and is largely determined by the digital processing workload. AP detection represents the dominant contributor due to its high computational intensity and its impact on duty cycle. The targeted improvements therefore prioritize reducing the energy cost of AP detection. At the SoC scale, the objective is to bring processing-related power consumption down to the same order of magnitude as acquisition-related power, i.e., into the μW range.

A second objective is to increase the number of templates that can be used for template matching while honoring the hard real-time constraint imposed by buffered acquisition. In the baseline system, processing must complete within the 500 ms buffer interval, yet the reported execution time exceeds this budget when more than one template is used. Since using multiple templates is desirable to improve the robustness of AP extraction, the targeted improvements aim to enable multi-template operation without violating the per-buffer timing budget. Practically, this requires reducing the per-template cost of NCC evaluation.

A third objective is to preserve, and ideally improve, the seizure-related information captured by the algorithm. In the present work, performance is assessed primarily through the detection metric $m[n]$ and the associated metric score MSC rather than through binary classification accuracy, as discussed in Section 1.3.3. The targeted modifications should therefore maintain or increase MSC relative to the baseline while avoiding degradations in the stability of $m[n]$ during baseline periods and in its sensitivity to seizure-related changes. This target constrains algorithmic simplifications and motivates systematic validation of each modification against the reference pipeline.

1.3.2 NCC hardware acceleration

The NCC evaluation performed during template matching constitutes the dominant computational workload of the AP detection stage and therefore directly drives both execution time and energy consumption in the embedded pipeline. In the baseline implementation, extending template matching to multiple templates increases the AP detection runtime beyond the per-buffer budget, which prevents the use of richer template sets. Consequently, accelerating NCC is identified as a high-leverage optimization point.

Hardware acceleration of NCC is adopted in this work as a mean to address two key performance targets simultaneously. First the active processing time per buffer can be reduced by offloading the most intensive operations of template matching

to dedicated hardware, enabling multi-template operation while meeting the hard real-time constraint. Second, the same offloading reduces the energy spent in AP detection, since the accelerated datapath can execute the required arithmetic with higher energy efficiency than a general-purpose MCU core. Importantly, this approach aims to conserve algorithmic behavior by preserving the NCC-based detection principle and its decision logic.

The design, integration, and evaluation of the NCC accelerator constitute the main contribution of this work. The proposed accelerator is developed to support the fixed-point embedded setting and the streaming nature of continuous acquisition and to interface with the existing processing chain such that the remaining steps can be retained with minimal modification.

1.3.3 Algorithm validation

In the present work, the algorithm is not evaluated through the binary classification output. The available experimental dataset contains too few seizure onsets to support a statistically meaningful assessment of classification performance. Instead, performance is quantified using the detection metric $m[n]$. To compare the baseline algorithm with subsequent refinements, a metric score MSC is defined. This score is the ratio between the RMS value of $m[n]$, evaluated at discrete analysis times t_n , over a seizure interval $\mathcal{S}$ and a baseline interval $\mathcal{B}$, as given in (1.9).

$$\text{MSC} = \frac{\sqrt{\frac{1}{N_s} \sum_{n: t_n \in \mathcal{S}} m^2[n]}}{\sqrt{\frac{1}{N_b} \sum_{n: t_n \in \mathcal{B}} m^2[n]}} \quad (1.9)$$

where $N_s = |\{n \mid t_n \in \mathcal{S}\}|$ and $N_b = |\{n \mid t_n \in \mathcal{B}\}|$. An example of the detection metric evolution with the seizure interval $\mathcal{S}$ highlighted and the corresponding MSC value indicated is shown in Figure 1.3. Higher values of MSC indicate a stronger increase of m during seizure activity relative to baseline, with $\text{MSC} > 1$ corresponding to an elevated RMS level during the seizure interval.

To run the seizure detection algorithm and evaluate its performance, a dataset collected through a collaboration between the *Cliniques Universitaires Saint-Luc* and the *Université Libre de Bruxelles* is used. The dataset contains six VENG recordings from rats in which seizures are induced by intravenous administration of pentylenetetrazol (PTZ), while the animals are concurrently monitored by electroencephalography (EEG). The recordings include temporal annotations defined by epileptologists, which segment each experiment into baseline (I0), preictal (I1),

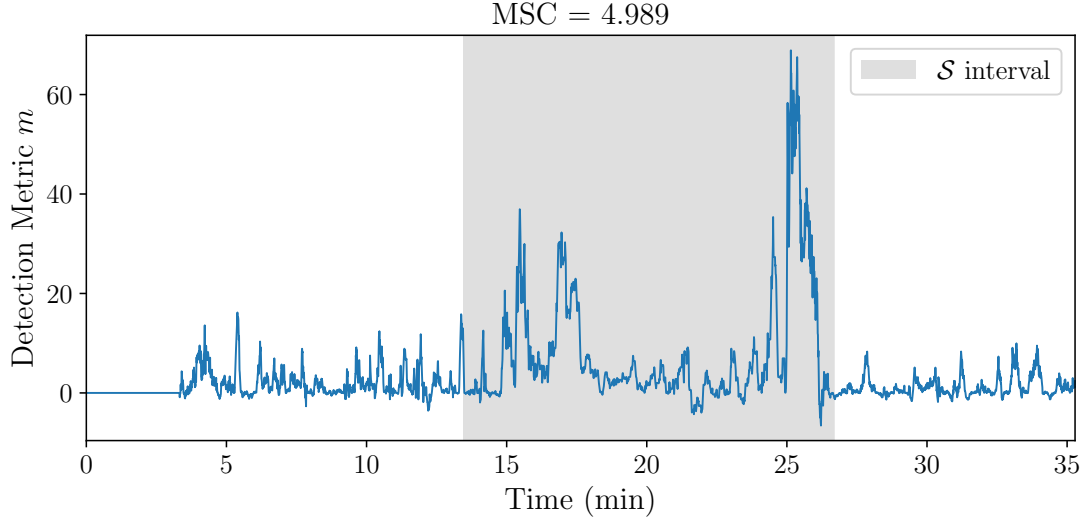


Figure 1.3: Example of the detection metric $m[n]$ evaluated over an entire recording. The shaded region indicates the seizure interval $\mathcal{S}$, while the remaining samples belong to the baseline interval $\mathcal{B}$. The metric score MSC is computed as the ratio of the RMS value of m within $\mathcal{S}$ to that within $\mathcal{B}$, yielding $\text{MSC} = 4.989$ for this example.

ictal (I2–I3), postictal EEG depression (I4), and recovery (I5) stages. More details on the experimental protocol and the annotation procedure are provided in [7], [5], [12], [13]. This experimental dataset is distinct from the one used in [6].

Each recording consists of two VENG channels sampled at 80 kS/s for approximately 30 minutes. The channels are combined into a single differential signal, defined as $s = s_1 - s_2$, where s_1 and s_2 denote the first and second channels respectively. An example of the resulting VENG trace is shown in Figure 1.4. As explained before, evaluation relies on the seizure interval $\mathcal{S}$ and the baseline interval $\mathcal{B}$ that account for the background window of duration T_{bg} used by the detection metric (see Section 1.1.2). Consequently, the seizure interval is defined as $\mathcal{S} = [t_{I1}, t_{I5} + T_{bg})$, where t_{Ik} denotes the annotated start time of interval Ik . The baseline interval $\mathcal{B}$ is defined as the complement of $\mathcal{S}$ over the recording duration.

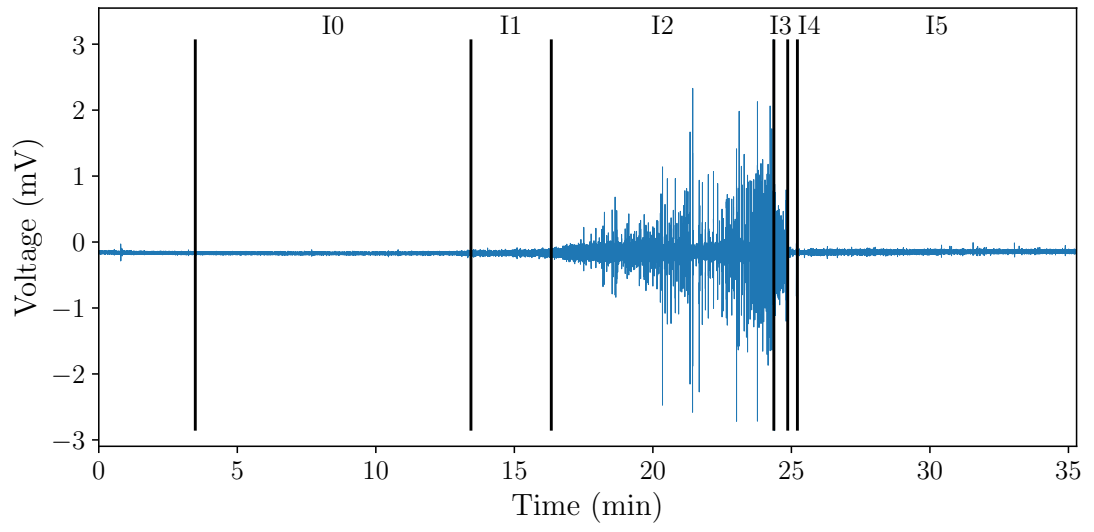


Figure 1.4: Example of a raw VENG recording from a rat during a PTZ-induced seizure experiment. Solid vertical lines indicate the annotated interval boundaries (t_{I0} – t_{I5}). According to the annotations, I1 is preictal, I2–I3 are ictal, I4 corresponds to postictal EEG depression, and I5 to recovery.

Chapter 2

Algorithm and software optimization

This second chapter presents the optimization of the VENG-based seizure detection algorithm and its implementation, with the objective of enabling an ultra-low-power, hardware-accelerated realization. Building on the baseline approach introduced in Chapter 1, optimization is addressed at both the algorithmic and implementation levels to reduce computational complexity, improve reliability, and enable a streaming, single-pass dataflow.

Algorithmic modifications are first introduced, including the removal of clustering, the normalization of templates, and the use of a correlation-based proxy for AP amplitude, leading to a consolidated optimized pipeline. The chapter then focuses on low-power implementation aspects, where platform assumptions are stated, two execution models are described, and word-length reduction strategies are applied. Finally, the implications of these choices for the design of a dedicated hardware accelerator are discussed, providing the basis for the accelerator architecture presented in the following chapter.

2.1 Algorithmic optimization

As detailed in Chapter 1, the seizure detection algorithm is organized into an AP detection stage followed by a seizure decision stage. In the baseline pipeline, successive 500 ms windows of the VENG signal are processed to extract AP features, namely the AP amplitude and the number of APs per window. The cornerstone operation is the normalized cross-correlation (NCC) between the incoming signal

and a set of AP templates, which produces a similarity trace used for event detection. Because NCC dominates the computational cost of the pipeline, it also represents the primary leverage point for achieving an ultra-low-power implementation.

AP candidates are obtained from the similarity trace using a fixed threshold combined with a simple distance rule to prevent multiple detections of the same event. Candidate events are then validated using an adaptive amplitude region defined relative to the signal RMS value. For each validated AP, the estimated amplitude is retained and propagated to subsequent processing. In parallel, a clustering mechanism is employed as a refinement step for the AP template set.

The seizure detection logic operates on the resulting AP feature stream to construct a metric reflecting changes in physiological activity. Foreground and background regions are maintained, and the mean absolute AP amplitude within each region is computed together with the AP frequency. Relative changes in amplitude and frequency between the regions are then derived, normalized, and combined into the final detection metric. Figure 1.2 illustrates this two-stage algorithm.

This section introduces a set of modifications to the baseline pipeline that will lay the foundation for the development of a hardware-accelerated NCC module. First, the clustering step is removed and replaced by a fixed pool of templates, enabling a deterministic formulation. The templates are normalized to unit norm, which simplifies the NCC computation and facilitates efficient hardware implementation. Building on this formulation, a key pipeline modification is proposed and validated to streamline processing and to enable a single-pass algorithm, in which the real-time signal is consumed only once by the NCC engine and is not reused elsewhere in the pipeline.

2.1.1 Removal of the clustering step

Template refinement via clustering aims to improve adaptability without requiring prior knowledge of AP morphologies. However, because clustering is an unsupervised and data-dependent procedure, the resulting template set may vary with the recent population of detected events. This variability reduces determinism and complicates verification and validation of the overall detection chain. In the context of regulated medical devices, current regulatory guidance for AI/ML-enabled software emphasizes lifecycle controls and highlights the importance of transparency, including explainability where applicable, as well as controlled change management to ensure continued safety and effectiveness over time [14], [15], [16]. For these reasons, and based on informal discussions with clinicians and medical

researchers involved in the project, clustering is considered a less suitable refinement mechanism from a system-integration perspective.

Instead, a fixed pool of twelve templates is employed. The templates are generated using the generic template generation tool provided in [17]. A width parameter is used to generate morphologically diverse templates while maintaining a constant length M of 40 samples. Since band-pass filtering constitutes the first stage of the AP detection algorithm, the templates are passed through the same second-order band-pass filter as the incoming VENG signal prior to NCC computation. In this work, the resulting template bank is defined offline and remains fixed during operation.

2.1.2 Template normalization

In (1.1), the cross-correlation between the length- M filtered signal segment $s[n + \tau]$ and the template $p_i[n]$ is normalized by the Euclidean norms of both sequences. Since clustering is removed and the template set is fixed during operation, the template norm can be absorbed offline by defining a unit-norm template

$$\bar{p}_i[n] = \frac{p_i[n]}{\|p_i\|_2}, \quad \|p_i\|_2 = \sqrt{\sum_{n=0}^{M-1} p_i^2[n]}. \quad (2.1)$$

Substituting $\bar{p}_i[n]$ into (1.1) yields an equivalent formulation in which only the signal-segment norm remains in the denominator:

$$\text{NCC}_i[\tau] = \frac{\sum_{n=0}^{M-1} s[n + \tau] \bar{p}_i[n]}{\sqrt{\sum_{n=0}^{M-1} s^2[n + \tau]}}. \quad (2.2)$$

This normalization reduces computational cost by eliminating the per-template normalization factor at runtime. Template norms therefore no longer need to be computed, stored, or applied. In the context of a hardware accelerator, unit-norm templates simplify coefficient scaling and remove one scaling operation per template, while preserving the numerical range of the similarity output. Figure 2.1 shows the resulting bank of normalized templates.

2.1.3 Correlation-based proxy for AP amplitude

The unit-norm template formulation introduced in the previous subsection not only simplifies NCC computation, but also exposes an intermediate quantity that can be

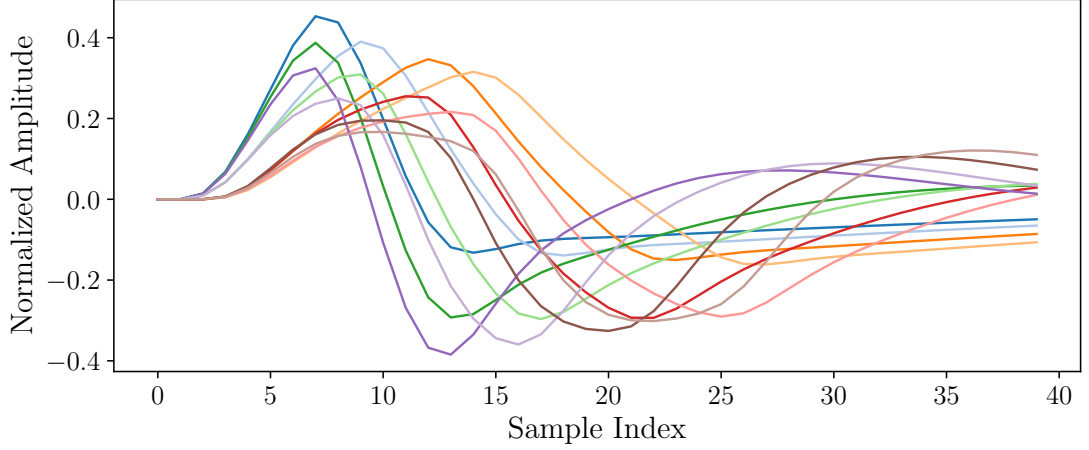


Figure 2.1: Bank of unit-norm AP templates used for NCC-based AP detection.

exploited to estimate AP amplitude. In the baseline pipeline, AP validation requires an amplitude estimate obtained from the time-domain signal, which typically implies retaining portions of the real-time signal beyond the NCC computation.

To streamline the dataflow, AP amplitude is instead inferred from the unnormalized correlation, i.e., the dot product, computed as part of the NCC operation. Denoting by

$$\text{Corr}_i[\tau] = \sum_{n=0}^{M-1} s[n + \tau] \bar{p}_i[n] \quad (2.3)$$

the correlation between the filtered signal segment and the unit-norm template $\bar{p}_i[n]$, a simple analytical intuition can be derived by modeling an AP occurrence at the detection lag τ^* as a scaled template,

$$s[n + \tau^*] \approx a \bar{p}_i[n] + \eta[n], \quad (2.4)$$

where a is the AP amplitude and $\eta[n]$ accounts for noise and modeling mismatch.

Substituting (2.4) into (2.3) yields

$$\text{Corr}_i[\tau^*] \approx a \|\bar{p}_i\|_2^2 + \sum_{n=0}^{M-1} \eta[n] \bar{p}_i[n] = a + \epsilon, \quad (2.5)$$

since $\|\bar{p}_i\|_2 = 1$. Therefore, at the detection lag τ^* , the correlation output $\text{Corr}_i[\tau^*]$ provides a direct proxy for the AP amplitude up to a bounded projection error ϵ .

Crucially, $\text{Corr}_i[\tau]$ is already produced by the NCC computation and can be made available as an accelerator output. As a result, the filtered real-time signal

only needs to be streamed into the hardware NCC engine and does not need to be stored in, nor reread from, a memory buffer for amplitude extraction, enabling a single-pass formulation of the AP detection stage.

2.1.4 Final optimized pipeline

Figure 2.2 summarizes the final optimized detection chain obtained by combining the previous modifications. For clarity, the complete pipeline is described below.

AP detection stage

1. The raw VENG signal is filtered by a second-order band-pass filter, which constitutes the first stage of AP detection. This filtering stage is assumed available in the intended embedded system and is discussed in the next section.
2. The filtered signal is processed by the NCC computation module. For each template, an NCC trace and its corresponding correlation trace are computed. Beyond this point, the time-domain signal does not need to be stored or reread for feature extraction.
3. The absolute values of the NCC traces are aggregated into a single similarity trace by selecting, at each sample, the maximum across templates. The aggregated correlation trace is formed consistently by selecting the correlation value associated with the same template index that maximizes the absolute NCC.
4. In non-overlapping 500 ms windows, candidate APs are identified as peaks of the similarity trace exceeding a threshold of 0.8 and separated by a minimum distance of 42 samples.
5. AP amplitudes are extracted from the aggregated correlation trace at the detected peak locations. Candidates are validated using an adaptive acceptance region based on the RMS of the aggregated correlation trace. An AP is accepted if its absolute amplitude is above the RMS and below four times the RMS.

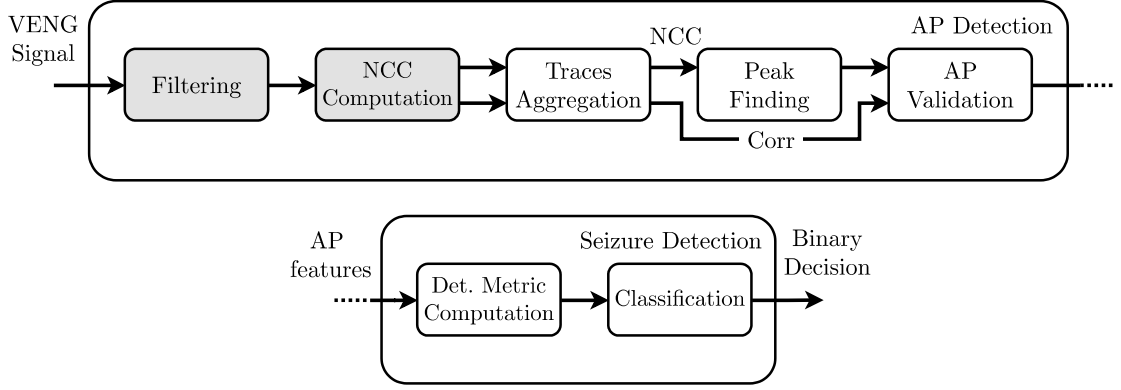


Figure 2.2: Optimized processing pipeline. Shaded blocks indicate operations intended for hardware acceleration. The VENG signal is filtered and processed by the NCC computation block. Template-relative traces are aggregated, candidate APs are detected from the similarity trace, and AP amplitudes are validated using the aggregated correlation trace as a proxy. The resulting AP feature stream is then processed by the seizure detection stage, yielding a binary decision on the patient state.

Seizure detection stage

1. The detection metric is computed using a foreground region of 20 windows and a background region of 180 windows, corresponding to 10 s and 90 s, respectively. The first 100 s are required to fill the foreground and background buffers. An additional 100 s are used to stabilize the estimated statistics prior to normalization. As a result, normalization of the relative amplitude and relative frequency becomes effective after a total warm-up duration of 200 s.
2. The final stage produces a binary decision from the detection metric, asserting the presence of a seizure.

2.2 Low-power implementation

This section translates the optimized algorithmic formulation into an implementation compatible with an ultra-low-power embedded deployment. First, the intended target platform is described and the resulting interface assumptions are stated, notably the availability of an analog front-end and a hardware band-pass filtering stage. The available dataset is then preprocessed to match the platform operating

conditions. Next, two software realizations of the pipeline are presented: an optimized, fully software-based reference implementation used for comparison against prior work, and a hardware/software-compatible version designed to follow the streaming execution model of the accelerator. Finally, the adopted word-length reduction strategy and the associated scaling choices are detailed.

2.2.1 Integration with the targeted platform

In [8], the BREWER chip is presented. It integrates an ultra-low-power dual-channel AFE tailored for VENG acquisition at 160 kS/s in an implantable system. An on-chip digital back-end (DBE) implements an IIR decimation filter that performs band-pass filtering from 210 Hz to 10 kHz while reducing the sampling rate to 20 kS/s. BREWER also embeds a Cortex-M4 MCU. This work uses BREWER as a reference deployment platform, and the proposed implementation is meant to meet with its interface and signal chain specifications.

As mentioned previously, band-pass filtering is available on the targeted platform. Although the DBE is configured for a 210 Hz–10 kHz band in [8], the filter parameters are reconfigurable and can be adjusted to match the intended 300 Hz–3 kHz band. Consequently, filtering is removed from the online algorithmic pipeline and emulated offline during dataset preprocessing. As described in Section 1.3.3, the available dataset is sampled at 80 kS/s. The preprocessing therefore includes a second-order Butterworth band-pass filter from 300 Hz to 3 kHz at 80 kS/s and subsequent resampling of the filtered signal to 20 kS/s using an FFT-based method, thereby matching the sampling rate assumed by the embedded implementation.

2.2.2 Software implementations and real-time scheduling

Two implementations of the optimized pipeline are considered. The first one is a fully software-based realization in which the NCC computation is implemented in software. Its purpose is to provide a fair basis for comparison with the baseline work in [6] in terms of runtime, power consumption, and memory requirements. As in the baseline approach, processing is performed every 500 ms on a buffered segment of filtered signal.

The second implementation is designed to be compatible with the proposed hardware accelerator and its streaming output interface. In this case, trace aggregation is executed each time a new accelerator output becomes available, while the remaining processing is carried out on 500 ms windows of the aggregated NCC and

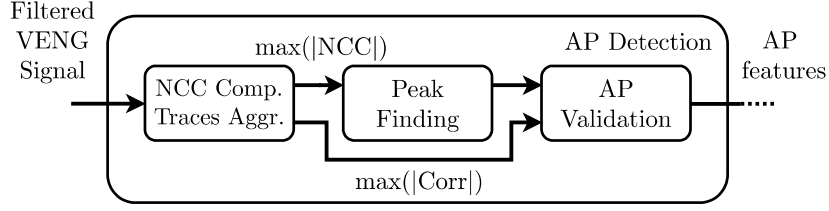


Figure 2.3: Reference implementation pipeline. The filtered signal is processed in buffered segments of 500 ms. For each buffer, NCC computation and trace aggregation are performed in software. Candidate APs are detected as peaks of the aggregated NCC trace and are validated using the aggregated correlation trace. The resulting AP features are forwarded to the seizure detection stage.

correlation traces. Details of the two implementations are provided below.

Reference implementation In the first implementation, NCC computation is carried out entirely in software using the C language. For each 500 ms signal segment, the signal norm is computed prior to correlation. To this end, the signal samples are squared and stored in a dedicated buffer, which is subsequently reused to accumulate the norm. This accumulation is implemented as a rolling sum, where a scratch variable holds the next squared sample to be subtracted from the running total as the window advances.

Correlation is then computed using the `arm_correlate_q31` function from the CMSIS-DSP library. For a signal segment of length $L = 10\,000$ samples, this function produces an output of length $2L - 1$, although only $L - M + 1$ samples correspond to the valid correlation range. The absolute value of this valid portion is extracted, after which the absolute NCC values are computed and used to update the aggregated NCC and correlation traces.

Overall, the software NCC implementation relies on four main working buffers: a correlation buffer of length $2L - 1$, a squared signal/norm buffer of length L , and two output buffers of length $L - M + 1$ storing the aggregated NCC and correlation traces. In addition, two signal buffers of length L are employed using a double-buffering scheme to decouple signal acquisition from processing.

The remaining steps reflect the algorithmic description given previously. The pipeline is executed once per 500 ms buffer of filtered signal. Consequently, the computation must complete within this 500 ms interval, which constitutes the sole hard real-time constraint of this reference implementation. Figure 2.3 illustrates the AP detection stage of the reference implementation pipeline.

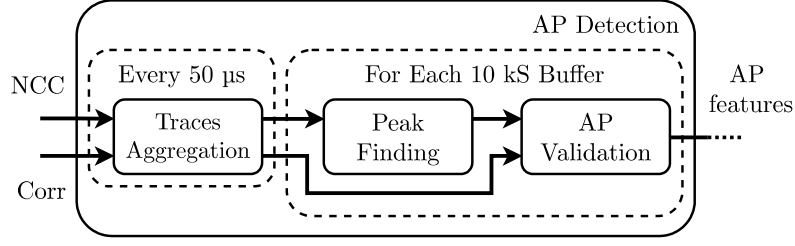


Figure 2.4: HW/SW-compatible implementation pipeline. At the sample rate (every 50 μ s), the accelerator outputs are consumed to incrementally build the aggregated NCC and correlation traces. Once 10 000 samples (500 ms at 20 kHz) have been accumulated, candidate APs are detected and validated. The resulting AP feature stream is then forwarded to the seizure detection stage.

HW/SW-compatible implementation In the HW/SW-compatible implementation, the results of the NCC computation are made available to the trace aggregation stage at the sampling rate. At each sampling instant, i.e., every 50 μ s, the NCC coefficients produced for the different templates are compared in software, and the template yielding the maximum absolute NCC value is selected. The absolute NCC value corresponding to this template is then appended to the aggregated NCC trace. Using the same template index, the associated absolute correlation coefficient is selected and appended to the aggregated correlation trace.

Once the aggregated NCC and correlation trace buffers have accumulated 10 000 samples, corresponding to a 500 ms interval at 20 kHz, the remaining stages of the pipeline are executed. To support continuous operation, a double-buffering scheme is employed. While one set of trace buffers is processed by the main pipeline, aggregation proceeds concurrently on a second set of buffers.

Because trace aggregation is triggered every 50 μ s, the aggregation routine must complete strictly within this interval while leaving sufficient execution margin for the rest of the system. This requirement constitutes the first hard real-time constraint of the HW/SW-compatible implementation and therefore mandates the highest execution priority to preempt any concurrent tasks.

In addition, the main processing pipeline is invoked every 500 ms and must execute within the time available between successive traces aggregation calls. More precisely, its execution time must remain below

$$0.5 - 10\,000 \times T_{\text{Agg}}. \quad (2.6)$$

seconds, where T_{Agg} denotes the execution time of a single aggregation step. Together with the aggregation constraint, this bound defines the real-time scheduling

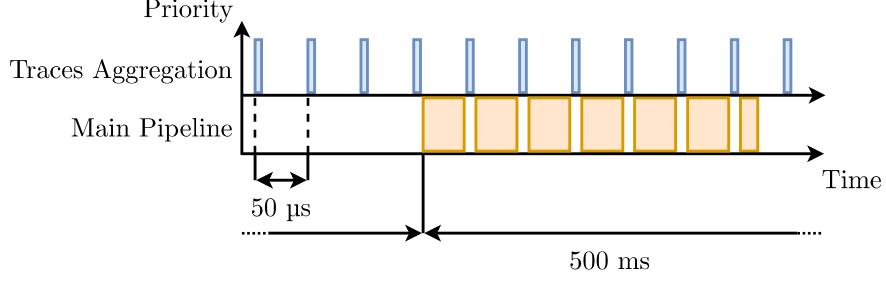


Figure 2.5: HW/SW-compatible implementation timeline. The trace aggregation task is executed every $50\ \mu\text{s}$ and preempts all other tasks. Its execution time must therefore remain strictly bounded within this interval. The main processing pipeline is executed every $500\ \text{ms}$ and must complete within the time available between successive aggregation calls.

requirements of the HW/SW-compatible pipeline. These constraints are summarized by the execution timeline shown in Figure 2.5. The AP detection stage of the corresponding processing pipeline is depicted in Figure 2.4.

2.2.3 Word-length reduction

To further reduce computational cost and memory footprint, word-length reduction is investigated for the optimized pipeline. Two software configurations are considered for the reference implementation: a fully 32-bit fixed-point version and a reduced-precision version relying primarily on 16-bit arithmetic. The latter configuration is also representative of the numerical format targeted by the hardware accelerator.

In the reduced-precision configuration, the input signal is converted from 32-bit to 16-bit fixed-point representation. Prior to conversion, a constant left shift of 10 bits is applied to the input samples. This scaling is selected empirically to maximize the effective precision of the NCC output for the provided dataset while minimizing saturation. The AP templates are converted to 16-bit representation without additional scaling, as their amplitude is already bounded by the unit-norm normalization described previously.

The NCC computation is then performed entirely in **q15** format. The signal norm is computed using **q15** buffers, and the correlation operation relies on the optimized `arm_correlate_opt_q15` function from the CMSIS-DSP library. Internally, this function uses a 64-bit accumulator to preserve numerical accuracy during accumulation. The correlation result is subsequently saturated back to **q15** format. Trace aggregation is performed using **q15** buffers for both the NCC and

the correlation traces. All subsequent steps of the AP detection stage following trace aggregation remain in `q15` format. In particular, the computation of the RMS value used for AP amplitude validation is performed using the `arm_rms_q15` CMSIS-DSP function, which also relies on a 64-bit accumulator internally to limit precision loss.

In the seizure detection stage, AP amplitudes are accumulated over foreground and background regions using `q31` buffers to preserve sufficient dynamic range. The final detection metric is likewise represented in `q31` format. This mixed-precision strategy ensures that computationally intensive and memory-dominant stages benefit from reduced word length, while accumulation-heavy operations retain adequate numerical robustness.

The reduced-precision configuration described above is the one adopted in the HW/SW-compatible implementation presented in the previous subsection. In this configuration, all signals exchanged between the software and the accelerator are represented using 16-bit fixed-point formats, and the internal accelerator data path is likewise designed around 16-bit input and output precision. Aligning the software numerical representation with the accelerator interface ensures consistency between the two implementations and enables a fair assessment of performance, power, and memory trade-offs when transitioning from the reference software execution to the hardware-accelerated pipeline.

2.3 Implications for the hardware accelerator

The algorithmic and implementation-level optimizations introduced in this chapter directly inform the architectural requirements of the hardware accelerator developed in the next chapter. In particular, they define the data formats, throughput constraints, and interface expectations that the accelerator must satisfy to integrate seamlessly with the optimized detection pipeline.

First, the accelerator is required to process real-time input samples produced at a sampling frequency of 20 kS/s. Consistent with the reduced-precision configuration adopted in the HW/SW-compatible implementation, all input and output signals of the accelerator are represented using 16-bit fixed-point formats. The accelerator must compute, for each template, both the correlation and the corresponding normalized cross-correlation coefficient on a per-sample basis using streamed input data.

Second, while the accelerator input and output formats are limited to 16-bit precision, sufficient internal precision must be provided for accumulation operations

to ensure numerical behavior comparable to the reduced-precision software reference. In particular, correlation must rely on wider internal data paths, followed by appropriate saturation back to q15. This requirement is essential to preserve detection performance while benefiting from word-length reduction at the interfaces.

Third, the accelerator must internally store the AP template bank and support correlation with multiple templates in parallel or quasi-parallel fashion. Although the present work relies on a fixed template bank defined offline, future versions of the algorithm may reintroduce template refinement mechanisms. Consequently, the accelerator architecture should allow for template updates through a programmable interface, without requiring full hardware redesign.

Fourth, the accelerator must expose both the NCC coefficients and the raw correlation results for each template at its output. As shown in Chapter 2, the NCC coefficients are required for candidate AP detection, while the correlation results are exploited as a proxy for AP amplitude during validation. Providing access to both quantities is therefore a fundamental requirement imposed by the optimized pipeline and its single-pass dataflow.

Finally, the accelerator is intended to operate in close cooperation with the Cortex-M4 core embedded in the BREWER platform, exchanging data through a shared system bus and control interface. Such an integration would enable the accelerator to act as a tightly coupled coprocessor, providing high-rate correlation outputs to the MCU while relying on software for aggregation, validation, and decision logic.

However, the concrete realization of this HW/SW integration, covering aspects such as bus protocols, register mapping, arbitration, and interrupt handling, is considered out of scope for the present work. Instead, the focus is placed on defining the accelerator functionality, data formats, and throughput requirements required by the optimized detection pipeline. The architectural design presented in the next chapter therefore assumes the existence of a suitable integration infrastructure and concentrates on the internal structure of the accelerator itself.

These considerations serve as the basis for the hardware accelerator architecture presented in the following chapter.

Chapter 3

Design and implementation of a normalized cross-correlation hardware accelerator

In this chapter, a hardware accelerator for real-time NCC with VENG signals is designed and implemented. After defining the system context and the design constraints, several architectural solutions are explored and compared using a first-order cost model. Based on this analysis, the parallel-templates architecture is selected as the most suitable candidate.

The detailed microarchitecture of the selected design is then described, including the organization of the memory, control, and computational datapaths. Finally, the verification methodology and implementation toolflow are introduced to validate the functionality and evaluate the performance and numerical accuracy of the proposed accelerator.

3.1 Design assumptions and accelerator interface

This section establishes the design assumptions and the HW/SW interface expectations used throughout the hardware accelerator development. First, the specific normalized cross-correlation formulation implemented in hardware is restated to fix notation for the remainder of the chapter. Then, the design constraints derived from the algorithmic implications identified in Section 2.3 are formalized. The remaining degrees of freedom are subsequently exploited to explore alternative architectures and select the one most suited to the targeted operating point.

Let $s[n]$ denote the band-pass filtered VENG signal at the input of the AP detection chain, and let $\bar{p}_i[n]$ be the i -th unit-norm template of length M samples. At each lag τ , the accelerator computes the correlation between the current M -sample signal segment and each template,

$$\text{Corr}_i[\tau] = \sum_{n=0}^{M-1} s[n + \tau] \bar{p}_i[n], \quad (3.1)$$

together with the corresponding signal-segment energy

$$E[\tau] = \sum_{n=0}^{M-1} s^2[n + \tau]. \quad (3.2)$$

Using these quantities, the normalized cross-correlation coefficient is defined as

$$\text{NCC}_i[\tau] = \frac{\text{Corr}_i[\tau]}{\sqrt{E[\tau]}}. \quad (3.3)$$

This formulation matches the unit-norm template reformulation introduced in Section 2.1: template normalization is performed offline, while only the signal-segment normalization term remains in the denominator. Importantly, the accelerator is not required to perform trace aggregation or template selection. These operations remain software responsibilities, as in the HW/SW-compatible pipeline described in Section 2.2.

The template length is fixed to $M = 40$ samples. While a larger template set is available at the algorithmic level, the number of templates simultaneously processed by the accelerator is limited to $N_T = 8$. This choice reduces the amount of data stored in the accelerator local memory (LMEM) and the associated datapath complexity. The considered accelerator architectures are designed to be scalable with respect to N_T and to support punctual updates of the active template bank.

The accelerator processes a real-time input stream at a sampling frequency of $f_s = 20$ kS/s, corresponding to one new input sample every 50 μ s. The computation is organized such that this input rate can be sustained without stalling the stream. Signed input samples are provided in 16-bit fixed-point format (**Q15**), where 15 bits represent the fractional part and the most significant bit encodes the two's complement sign. Internally, wider word-length representations are permitted for accumulation and intermediate computations in order to preserve numerical precision.

At the interface level, the accelerator outputs two sets of **Q15**-formatted quantities. For each template and each lag τ , both $\text{Corr}_i[\tau]$ and $\text{NCC}_i[\tau]$ are made

available. Providing both quantities avoids redundant computation in software and enables the correlation-based amplitude proxy described in Section 2.1. The exact output timing and latency depend on the selected architecture.

Finally, concrete integration with the on-chip MCU, covering aspects such as bus protocols, register mapping, DMA, and interrupt handling, is considered out of scope for this work. Accordingly, the accelerator is described using an abstract streaming interface, which is sufficient to support architecture exploration, implementation, and verification within the scope of this work.

3.2 Architecture exploration and selection

Based on the design assumptions and interface expectations established in the previous section, several hardware architectures can be considered to implement the NCC formulation. Although the mathematical operations defined in (3.1), (3.2) and (3.3) are fixed, multiple architectural strategies exist to map those computations onto hardware, differing in how parallelism is exploited across templates, time indices, and arithmetic operations.

This section explores a set of four candidate architectures. For each candidate, the streaming dataflow, buffering requirements, and compute structure are described, together with a first-order microarchitectural sketch. These architectures are then compared using simple cost models capturing computational effort, LMEM requirements, and memory access activity. Based on this comparative analysis, the architecture most suited to the targeted operating point and system constraints is selected and carried forward for detailed implementation.

3.2.1 Candidate architectures and dataflow

Parallel-templates

In this architecture, the correlation between the input signal and each of the N_T templates is computed in parallel, while the signal energy is accumulated concurrently. Once the correlations and the signal energy are available, a second stage computes the reciprocal square root (RSQRT) of the energy and forms the corresponding NCC coefficients sequentially. A first-order microarchitectural sketch of this approach is shown in Figure 3.1.

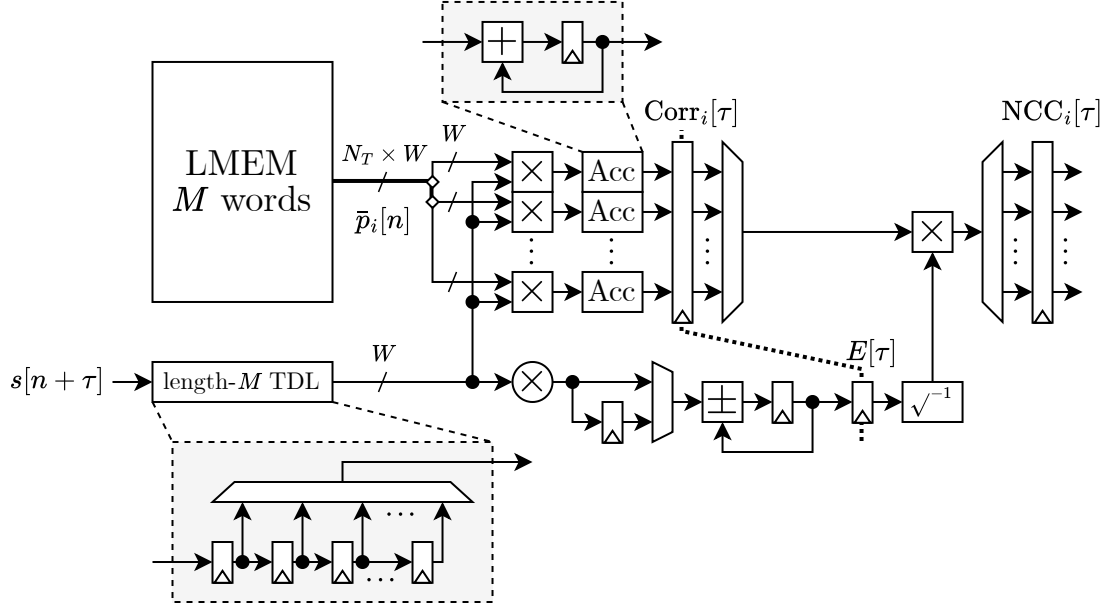


Figure 3.1: First-order microarchitectural sketch of the parallel-templates architecture. The input stream feeds a length- M TDL. Samples are then delivered sequentially to the correlation datapath (upper part) and to the signal energy accumulation datapath (lower part). Templates are stored on the LMEM in a column-wise fashion and are streamed row by row to the correlation datapath. Correlation with each template is performed in parallel, while the signal energy is updated using a rolling sum. In a second stage (right of the dotted line), the RSQRT of the energy is computed and can be multiplied sequentially with the correlation results to produce the NCC coefficients. The correlation and NCC results are made available on their respective registers.

The architecture is organized into two main stages. The first stage receives W -bit input samples and stores them in a tapped delay line (TDL) of length M , which maintains the sliding window of signal samples required for both correlation and energy computation. Templates are stored in the accelerator LMEM in a column-wise fashion and are streamed row by row to the first stage. The resulting $(N_T \times W)$ -bit output stream from the LMEM is split into N_T independent W -bit streams, isolating each template sample on its own dataline. At this point, two datapaths operate in parallel.

Correlation datapath The correlation datapath computes the correlation between the signal and each template in parallel. Specifically, N_T multipliers multiply the signal samples provided by the TDL with the corresponding template samples

read from the LMEM. The resulting products are accumulated to implement the required multiply-and-accumulate (MAC) operations. Because signal samples are streamed in natural order (newest sample first), the template samples are streamed in reverse order to obtain the correct correlation results. After M MAC operations, the correlation result for each template is registered and made available both to the second stage and at the accelerator output.

Signal energy datapath The signal energy datapath computes the signal energy using a rolling sum technique. In this datapath, only the newest and the oldest samples of the current signal window are involved in the computation, while a register maintains the accumulated energy value. When a new sample is streamed from the TDL, it is squared and added to the energy accumulator. Subsequently, the square of the oldest sample from the previously processed window is subtracted from the accumulated energy. To support this operation, a register stores the squared value of the oldest sample and is updated when the TDL outputs the oldest sample of the current window. As a result, three distinct operations are performed in sequence at each step: addition of the newest squared sample, subtraction of the previous oldest squared sample, and update of the stored oldest sample. The resulting energy value is then registered and made available to the second stage.

The second stage computes the RSQRT of the signal energy and subsequently forms the N_T NCC coefficients by multiplying the RSQRT result with the corresponding correlation values. This stage operates sequentially across templates. This design choice is motivated by the fact that the first stage performs M MAC operations per input sample, whereas the second stage requires only one RSQRT operation and N_T multiplications. Since M is larger than N_T plus the fixed number of operations required to compute the RSQRT (denoted N_{SQ}), the second stage does not limit throughput. As a result, a single multiplier can be reused to compute all N_T NCC coefficients. As with the correlation results, the NCC values are registered and made available at the accelerator output.

Parallel-lags

In this architecture, correlation is computed in parallel for J consecutive lags by evaluating J overlapping length- M signal windows against a given template. Adjacent windows are offset by one sample, so that the required signal samples are available to form J lagged correlations simultaneously. In contrast, the corresponding signal energies are accumulated sequentially using a rolling-sum update. Once the J correlation values and the J energy values are available, a second stage computes the RSQRT of each energy and forms the associated NCC coefficients

sequentially. A first-order microarchitectural sketch of this approach is shown in Figure 3.2.

As in the previous architecture, this one is organized into two main stages. The first stage receives W -bit input samples and stores them in a TDL of length $M + J - 1$ with $J + 1$ taps. The N_T templates are stored in the LMEM with one template sample per row, and templates are streamed to the datapath one template at a time. Processing is performed in blocks of J new input samples. Once J samples have been fed into the TDL, the architecture computes the J correlation values and the corresponding J NCC coefficients for each template in the active template bank. A processing block completes when these J correlation and J NCC results have been produced for all N_T templates. Similarly to the previous architecture, two datapaths operate in parallel.

Correlation datapath In this datapath, J MAC lanes operate in parallel. Each lane receives a shifted signal sample from one of the first J taps of the TDL and multiplies it with the corresponding template sample read from the LMEM. Template samples are streamed in reverse order to match the natural ordering of the signal samples and therefore produce the correct correlation values. After M MAC iterations per lane, the J correlation results are registered and made available both to the second stage and at the accelerator output.

Signal energy datapath To compute the energy of each shifted signal window, this datapath applies the same rolling-sum principle as described previously: an accumulator maintains the current energy estimate, and a companion register stores the squared sample that must be subtracted when the window advances. In the parallel-lags architecture, the key difference is that energy computation is performed sequentially across the J lags within each processing block. Concretely, after a block of J new input samples has been loaded into the TDL, the datapath iterates over $j = 0, \dots, J - 1$ and updates the corresponding energy $E[\tau + j]$ one lag at a time using the rolling-sum update. The last tap of the TDL is used to supply samples in the order required by this sequential schedule, independently of the ordering used in the correlation datapath. Each updated energy value is registered as soon as it becomes available, allowing the datapath to proceed immediately to the next lag update. After completing the J sequential updates, the set of J registered energy values is available to the second stage.

Since the signal energy is independent of the currently processed template in the correlation datapath, this energy computation only needs to be performed once per processing block during the processing of the first template. The resulting J energy values can then be reused for the remaining $N_T - 1$ templates.

To avoid stalling the correlation datapath, the total number of operations required to produce the J energy values must fit within the M -operation budget of the correlation accumulation. Under the rolling-sum schedule, three operations are performed per lag update: addition of the newest squared sample, subtraction of the previous oldest squared sample, and update of the stored oldest squared sample. This leads to the following first-order constraint:

$$3 \times J \leq M. \quad (3.4)$$

With $M = 40$, this implies $J \leq 13$.

The second stage computes the RSQRT of each of the J energy values and forms the corresponding NCC coefficients sequentially using a single multiplier. The RSQRT operation requires a fixed number of elementary operations, denoted N_{SQ} . In the parallel-templates architecture, this cost was incurred only once per lag and its scheduling could be neglected because the RSQRT result was reused across templates. In the parallel-lags architecture, however, an RSQRT must be computed for each of the J lags, and each RSQRT result must then be multiplied by the corresponding correlation value to produce the NCC coefficient.

To avoid stalling the first stage, the total work required by the second stage to produce the J NCC coefficients must fit within the M -operation budget associated with the correlation accumulation. Under a first-order model in which each NCC coefficient requires N_{SQ} operations for the RSQRT computation plus one multiplication, the following constraint is obtained:

$$(N_{\text{SQ}} + 1) J \leq M. \quad (3.5)$$

Finally, the J NCC values are registered as they are produced and made available at the accelerator output. It is worth noting that this architecture would require adaptations on the software side, since trace aggregation would need to operate on J consecutive correlation/NCC outputs per processing block.

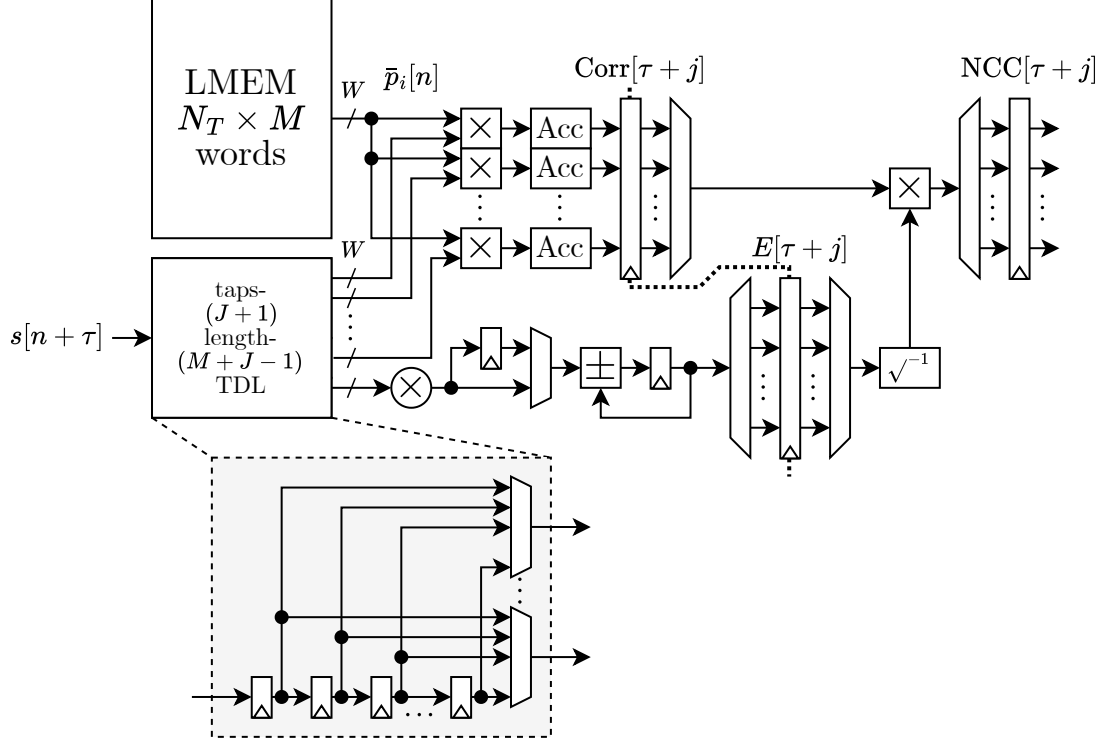


Figure 3.2: First-order microarchitectural sketch of the parallel-lags architecture. The input stream feeds a TDL with $J + 1$ taps and an overall length of $M + J - 1$. The first J taps feed the correlation datapath, while the last tap feeds the signal energy accumulation datapath. The N_T templates are stored in the LMEM, with one template sample per row, and are streamed to the correlation datapath one template at a time. Correlation for J consecutive lags is performed in parallel, whereas the corresponding J signal energies are computed sequentially using a rolling sum. In a second stage, the RSQRT of the j -th energy is computed and multiplied sequentially with the corresponding correlation result to produce the j -th NCC coefficient. This process is repeated for $j = 0, \dots, J - 1$. While J new input samples are fed into the TDL, the above computation is carried out sequentially for each of the N_T templates.

Bi-parallel

The bi-parallel architecture exploits parallelism along both the template and lag dimensions. For each of the N_T templates, correlations are computed for J consecutive lags. Conceptually, this architecture combines the core ideas of the parallel-templates and parallel-lags approaches. As in the parallel-lags architecture, the J lagged correlations are obtained from J shifted windows of the input signal, while the corresponding signal energies are updated sequentially using a rolling-sum mechanism. At the same time, the correlation datapath is replicated across templates: N_T parallel lanes perform the parallel-lags correlation computation concurrently, yielding $N_T \times J$ correlation values per processing block. In the second stage, the RSQRT of each of the J energy values is computed sequentially. For each lag j , the corresponding NCC coefficients are then formed for all templates in parallel by multiplying $\text{RSQRT}(E[\tau + j])$ with the correlation results. A first-order microarchitectural sketch of this approach is provided in Figure 3.3.

Input sample handling is identical to the parallel-lags architecture using a TDL of length $M + J - 1$ with $J + 1$ taps. The J first taps are fed to the correlation lanes, while the last tap is used by the signal energy datapath. The LMEM handles the templates just like in the parallel-templates architecture. The splitted N_T independent template sample streams are fed to the N_T correlation lanes.

Input sample handling follows the same approach as in the parallel-lags architecture. A TDL of length $M + J - 1$ with $J + 1$ taps is used to provide the J shifted signal windows required for lag-parallel correlation. The first J taps feed the N_T correlation lanes, while the last tap is used by the signal energy datapath. Template handling in LMEM follows the parallel-templates organization: templates are stored column-wise and streamed row by row. The resulting $(N_T \times W)$ -bit LMEM output is split into N_T independent W -bit streams, which are routed to the N_T correlation lanes.

Correlation datapath The correlation datapath comprises N_T template lanes operating in parallel. Within each template lane, J MAC lanes operate in parallel to compute correlations for J consecutive lags. Each MAC lane receives a shifted signal sample from the corresponding TDL tap and multiplies it by the current template sample read from the LMEM. Template samples are streamed in reverse order to match the natural ordering of the signal samples and thus produce the correct correlation values. After M MAC iterations, the $N_T \times J$ correlation results (one per template and per lag) are registered and made available both to the second stage and at the accelerator output.

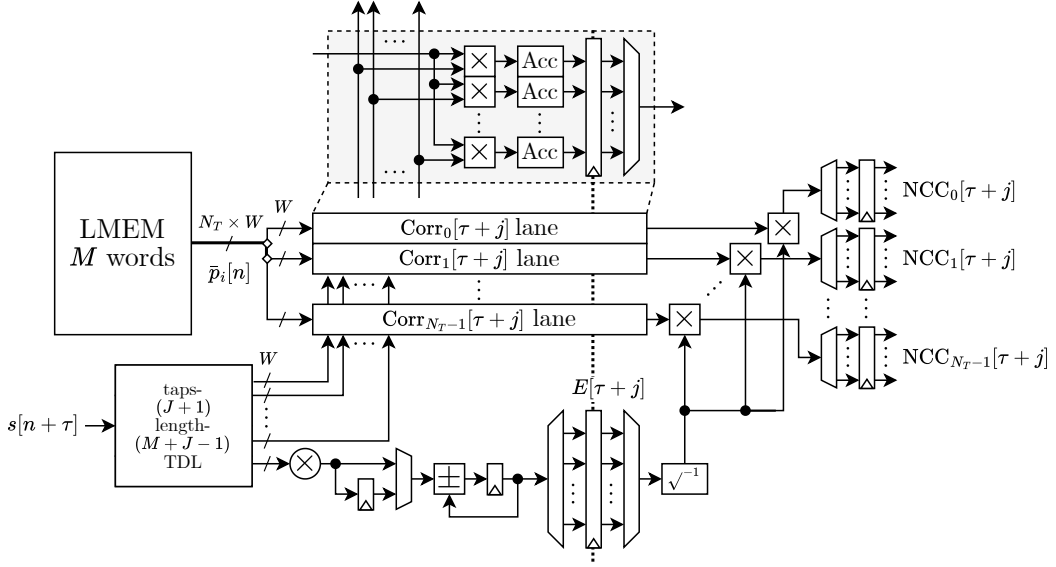


Figure 3.3: First-order microarchitectural sketch of the bi-parallel architecture. The input stream feeds a TDL with $J + 1$ taps and an overall length of $M + J - 1$. The first J taps feed the correlation datapath, while the last tap feeds the signal energy accumulation datapath. Templates are stored in the LMEM in a column-wise fashion and are streamed row by row to the correlation datapath. Correlation is computed in parallel across both dimensions: J consecutive lags and N_T templates, while the corresponding J signal energies are computed sequentially using a rolling sum. In a second stage, the RSQRT of the j -th energy is computed and, for all templates in parallel, the j -th NCC coefficient is formed. This process is repeated for $j = 0, \dots, J - 1$.

Signal energy datapath The signal energy datapath follows the same rolling-sum implementation as in the parallel-lags architecture. Since the signal energy is independent of the currently processed template, it is computed once per processing block and reused across all N_T template lanes. Consequently, the same first-order scheduling constraint derived for the parallel-lags case applies here, namely $3 \times J \leq M$, which yields $J \leq 13$ for $M = 40$.

In the second stage, the RSQRT of each of the J energy values is computed sequentially and the corresponding NCC coefficients are formed for all templates in parallel. Consequently, only a single RSQRT module is required, while N_T multipliers are instantiated to compute the N_T NCC values associated with a given lag j in parallel. As in the parallel-lags architecture, the second stage must satisfy

the first-order throughput constraint

$$(N_{\text{SQ}} + 1) J \leq M, \quad (3.6)$$

so that RSQRT computation and NCC formation for each of the J lags can be achieved without stalling the first stage. Finally, the resulting $N_T \times J$ NCC values are registered as they are produced and made available at the accelerator output. It is worth noting that this architecture would require adaptations on the software side, since trace aggregation would need to operate on a block of J consecutive lagged correlation/NCC outputs for all N_T templates at each processing block.

Parallel-indices

The parallel-indices architecture exploits parallelism across the summation indices of the correlation operation. Rather than computing the dot product through M sequential MAC steps, the M products are formed concurrently and reduced using an adder tree to obtain the correlation in a single pass for a given length- M signal window and template. In parallel, the signal energy is updated using a rolling-sum, after which its RSQRT is computed and used to form the corresponding NCC coefficient. This process is repeated for each of the N_T templates of the active template bank. A first-order microarchitectural sketch of this approach is shown in Figure 3.4.

Input samples are stored in a register bank of M registers using a write pointer that selects which register is updated at each new sample, such that the bank acts as a circular buffer holding the most recent length- M signal window. The M register outputs are exposed in parallel to both the correlation datapath and the signal energy datapath. The active template bank is stored in the LMEM in a row-wise fashion, with one full template per $(M \times W)$ -bit row. Because the register bank does not impose a fixed temporal ordering on its output lines (the position of the newest and oldest samples depends on the write pointer), the template samples must be permuted accordingly so that the multiplier inputs receive correctly aligned pairs $(s[n + \tau], \bar{p}_i[n])$. To this end, the $(M \times W)$ -bit LMEM output is split into M independent W -bit streams and routed through a permutation network controlled by the write pointer. The two datapaths described below then operate in parallel.

Correlation datapath The correlation datapath first forms the M products between the M signal samples and the M template samples in parallel. The template samples are permuted by the permutation network so that each multiplier receives correctly aligned pairs $(s[n + \tau], \bar{p}_i[n])$, despite the circular addressing of

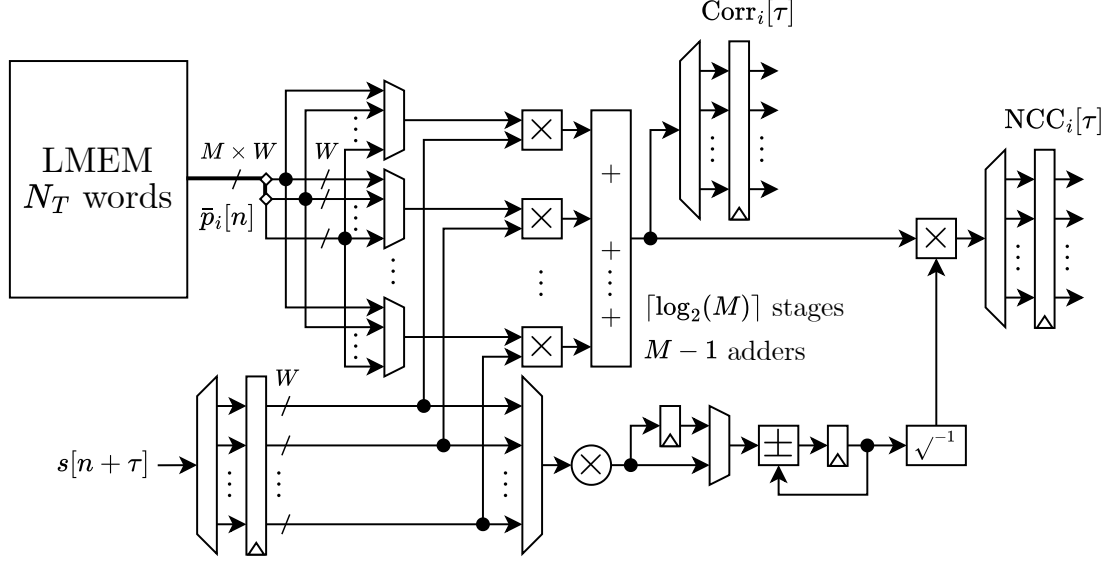


Figure 3.4: First-order microarchitectural sketch of the parallel-indices architecture. The input stream feeds a register bank of depth M that holds the current length- M signal window and exposes the M samples in parallel to the correlation datapath. Templates are stored in the LMEM in a row-wise fashion, with one full template per row. For a given template, the correlation between the current signal window and the template is computed in a single pass using M multipliers and an adder tree of $M - 1$ adders organized into $\lceil \log_2(M) \rceil$ stages. In parallel, the signal energy is updated using a rolling-sum mechanism and its RSQRT is computed. The resulting RSQRT value is template-independent and can therefore be reused to form the NCC coefficient associated with each template.

the register bank. The resulting products are then reduced using an adder tree. For $M = 40$, the reduction requires $\lceil \log_2(M) \rceil = 6$ adder stages and a total of $M - 1$ two-input adders. After the reduction, the correlation result is forwarded to the NCC formation multiplier and registered to be exposed at the accelerator output.

Signal energy datapath As in the previous architectures, the signal energy is computed using a rolling-sum update driven by an accumulator register and a companion register storing the squared sample to be subtracted when the window advances. At each new input sample, the required newest and oldest samples of the length- M window are selected from the circular register bank based on the write pointer, squared, and used to update the accumulated energy through the three standard operations (addition, subtraction, and update of the stored oldest square). Once the updated energy value is available, its RSQRT is computed.

Since the signal energy is template-independent, the RSQRT result is computed once per window update and reused across the N_T iterations of the correlation datapath required to process the active template bank. For each template, the corresponding NCC coefficient is then formed by multiplying the template correlation result by the shared RSQRT value. Finally, the NCC coefficient is registered and exposed at the accelerator output.

3.2.2 First-order cost model

To compare candidate architectures, a first-order cost model is used. The objective is not to predict absolute power consumption or exact resource counts, but to derive simple, consistent metrics that reflect real-time feasibility, compute footprint, and local-memory requirements at the targeted operating point. For a fair comparison of parallelization strategies, the degree of parallelism is fixed across candidates by setting the number of templates to $N_T = 8$ and, for lag-parallel architectures, the number of parallel lags to $J = 8$. The template and signal window length is fixed to $M = 40$, and inputs/outputs are represented using 16-bit fixed-point formats.

Real-time feasibility Real-time feasibility is primarily assessed through the cycle count per input sample (CPS), which captures the sustained throughput of each architecture under the adopted scheduling model. For a given accelerator clock frequency f_{clk} and input sampling frequency f_s , the available cycle budget per input sample is $C_{\text{budget}} = f_{\text{clk}}/f_s$. An architecture is considered compatible with the input stream if its CPS does not exceed this budget. In the following, one MAC operation is assumed to complete in a single clock cycle. For the parallel-indices architecture, the reduction tree is modeled at a first-order level by assuming that two adder-tree stages can be evaluated per clock cycle.

In addition, an output-service metric is reported to capture the burstiness of the produced outputs. Here, the burst size corresponds to the number of accelerator output values that become available over a short interval as a consequence of the architecture scheduling. For block-based architectures, this burst size increases proportionally to the block size J , since J consecutive lags are produced per processing block. Table 3.1 summarizes CPS and burst size for each candidate architecture.

Under the comparison settings ($M = 40$, $N_T = 8$, $J = 8$), the bi-parallel architecture achieves the lowest CPS and therefore provides the largest throughput margin. In particular, sustaining $f_s = 20$ kS/s requires $f_{\text{clk}} \geq \text{CPS} \cdot f_s$, which translates to a minimum clock frequency of 800 kHz for the parallel-templates

Architecture	CPS (cycles/sample)		Burst size (values)	
parallel-templates	40	M	16	$2N_T$
parallel-lags	40	MN_T/J	128	$2N_TJ$
bi-parallel	5	M/J	128	$2N_TJ$
parallel-indices	32	$(\lceil \log_2(M) \rceil / 2 + 1) N_T$	16	$2N_T$

Table 3.1: Real-time feasibility metrics for the four candidate architectures. CPS denotes the steady-state cycle count per input sample, while burst size denotes the number of output values (Corr and NCC) produced per scheduling block. For each metric, the left column reports the value obtained for the comparison settings ($M = 40$, $N_T = 8$, $J = 8$) and the right column provides the corresponding general expression.

and parallel-lags architectures, 640 kHz for the parallel-indices architecture, and 100 kHz for the bi-parallel architecture. As expected, the burst size is larger for the two block-based architectures (parallel-lags and bi-parallel) by a factor of J , since they produce $2N_TJ$ output values per processing block instead of $2N_T$.

Compute resources A first-order estimate of the compute footprint of each candidate architecture is obtained by counting the main arithmetic resources required, namely multipliers, accumulators, and adders in the adder-tree structure of the parallel-indices architecture. The RSQRT operation is implemented using a single shared module in all candidates. It is therefore not a differentiating factor in the comparison.

As shown in Table 3.2, the parallel-templates and parallel-lags architectures require a comparable and relatively small number of arithmetic units, since parallelism is exploited along a single dimension. In contrast, the bi-parallel architecture exhibits a sharp increase in both multipliers and accumulators due to the replication of correlation lanes across two dimensions. The parallel-indices architecture also incurs a substantial compute footprint, as full index parallelism requires M multipliers and an adder tree of $M - 1$ adders. Overall, these first-order counts suggest that the bi-parallel and parallel-indices candidates are less attractive from an area and switching-activity perspective despite their favorable CPS.

Local memory footprint The LMEM requirements constitute a key metric for architecture selection, as they directly influence area and energy consumption. To compare candidate architectures, the LMEM is characterized by its word width (in bits) and word depth (number of words). While several memory implementations

Architecture	Compute element count (No. of)					
	Multipliers		Accumulators		Adders (tree)	
parallel-templates	10	$N_T + 2$	9	$N_T + 1$	–	–
parallel-lags	10	$J + 2$	9	$J + 1$	–	–
bi-parallel	73	$(J + 1)N_T + 1$	65	$JN_T + 1$	–	–
parallel-indices	42	$M + 2$	1	–	39	$M - 1$

Table 3.2: First-order compute resource counts for each candidate architecture. For each entry, the left column reports the count obtained for the comparison settings ($M = 40$, $N_T = 8$, $J = 8$), while the right column provides the corresponding general expression. A single RSQRT module is assumed in all architectures and is therefore omitted from the table.

are possible, the most area- and energy-efficient choice depends on the target technology. In this work, the accelerator is intended for synthesis in a TSMC 65 nm technology node, where such LMEM instances are assumed to be implemented using SRAM macros.

A practical consequence of using SRAM macros is discretization: only a finite set of word widths and depths is available in the macro library used for this work. As a result, some architectures require either banking/replication of SRAM macros to reach the requested word width, or the use of a deeper macro than strictly necessary, which introduces unused memory capacity.

For example, the parallel-indices architecture requires a word width of $MW = 640$ bits. This exceeds the maximum word width available in the selected macro set, and therefore requires banking 5 SRAM instances. In addition, the required word depth of $N_T = 8$ words is below the minimum depth offered by the same macro set, resulting in a significant fraction of unused memory.

Beyond capacity, LMEM access activity is also reported, as it is a strong driver of dynamic energy. The read access rate (in reads/sample/template) is derived from the first-order schedules described previously and combined with a datasheet-provided per-read energy to obtain an energy proxy reported in pJ/V/sample/template. Since energy values are not provided for every possible (width, depth) combination, linear interpolation is used when required. Table 3.3 summarizes the LMEM width/depth requirements, read access rates, and the resulting normalized energy proxy for each candidate architecture.

Two dominant effects can be highlighted. First, the parallel-indices architecture is strongly penalized by LMEM shape constraints: its very wide word ($MW = 640$ bits) typically forces SRAM banking, while its shallow depth ($N_T = 8$ words)

Architecture	Word width (bits)		Word depth (words)		Access rate (reads/sample /template)		E_{proxy} (pJ/V /sample /template)
parallel-templates	128	$N_T W$	40	M	5	M/N_T	180.64
parallel-lags	16	W	320	$N_T M$	5	M/J	30.385
bi-parallel	128	$N_T W$	40	M	0.625	$M/N_T J$	22.58
parallel-indices	640	MW	8	N_T	1	–	898.38

Table 3.3: First-order LMEM requirements and access-activity proxies for each candidate architecture. For each metric, the left column reports the value obtained for the comparison settings ($M = 40$, $N_T = 8$, $J = 8$, $W = 16$), while the right column provides the general expression. The energy proxy is computed from the read access rate and datasheet-provided per-read energy values (normalized by voltage), using linear interpolation when required.

results in poor macro utilization; this combination is reflected by the largest normalized energy proxy. Second, for the remaining candidates, the normalized LMEM energy proxy is driven by the combination of per-read energy, which increases with word width, and the normalized read access rate. Parallel-lags achieves a low proxy due to its narrow word width, while parallel-templates is higher due to a wider word and a higher access rate. Bi-parallel achieves the lowest proxy (22.58 pJ/V/sample/template) because its read access rate is reduced by a factor $N_T J$, which more than compensates for its wider word width.

Beyond the LMEM macros used to store templates, additional on-chip storage is required to buffer the streaming input samples and to temporarily hold accelerator outputs until they are consumed by the rest of the system. This storage is modeled here as registers. Input buffering captures the storage needed to maintain the signal window used by the correlation and energy computations (TDL or register bank). Output buffering captures the storage needed to hold the produced output values (Corr and NCC) at the HW/SW boundary.

Table 3.4 shows that input buffering requirements are similar across candidates: the per-sample architectures (parallel-templates and parallel-indices) require M registers to store the current length- M window, whereas the lag-parallel architectures (parallel-lags and bi-parallel) require $M + J - 1$ registers to simultaneously expose J shifted windows. The dominant difference lies in output buffering. Block-based architectures require $2N_T J$ output registers, i.e., a factor J more than per-sample architectures. This increase directly reflects the bursty production of J consecutive lag results per processing block and translates into a higher storage and interface burden at the accelerator boundary.

Architecture	In/out buffering (No. of registers)			
	Input		Output values	
parallel-templates	40	M	16	$2N_T$
parallel-lags	47	$M + J - 1$	128	$2N_T J$
bi-parallel	47	$M + J - 1$	128	$2N_T J$
parallel-indices	40	M	16	$2N_T$

Table 3.4: First-order register storage requirements for input and output buffering. The input column corresponds to the number of registers required to hold the signal window (TDL/register bank), while the output column corresponds to the number of registers required to hold the accelerator output values (Corr and NCC) until they are consumed. For each metric, the left column reports the value obtained for the comparison settings ($M = 40$, $N_T = 8$, $J = 8$) and the right column provides the general expression.

3.2.3 Architecture comparison and selected design

The candidate architectures are now compared by jointly considering the metrics reported in the previous subsection. While the bi-parallel architecture exhibits the lowest CPS and therefore the largest throughput margin, real-time feasibility at $f_s = 20$ kS/s is not a limiting factor for the other candidates either, since they remain compatible with sub-MHz clock frequencies under the adopted scheduling assumptions. Instead, the burst size emerges as an important integration consideration: block-based architectures, i.e., parallel-lags and bi-parallel, produce $2N_T J$ output values per processing block, which increases output buffering requirements and complicates the HW/SW interface compared to per-sample architectures that expose $2N_T$ values.

From a compute-resource perspective, Table 3.2 clearly favors the parallel-templates and parallel-lags candidates. In contrast, the bi-parallel architecture incurs a large replication of arithmetic resources due to two-dimensional parallelism, and the parallel-indices architecture requires a wide dot-product datapath and adder-tree reduction, resulting in a substantially higher arithmetic footprint. These trends make bi-parallel and parallel-indices less attractive for an ultra-low-power objective, despite their favorable CPS.

Local memory considerations further differentiate the remaining candidates. The LMEM footprint and access-activity proxy in Table 3.3 tend to favor the parallel-lags organization due to its narrow word width and efficient access energy, whereas the parallel-templates organization requires wider words and therefore

higher per-read energy. However, Table 3.4 shows that the parallel-templates architecture benefits from substantially lower output buffering requirements: $2N_T$ output registers instead of $2N_TJ$ for block-based designs. Moreover, the LMEM access-energy gap between the two remaining candidates can be mitigated by leveraging register-file style SRAMs available in the target technology, which provide lower access energy for shallow memories and therefore reduce the penalty associated with wider words.

Overall, the parallel-templates architecture provides the most balanced trade-off between compute footprint, memory requirements, and interface simplicity at the targeted operating point. Consequently, it is selected for detailed microarchitectural design and implementation in the remainder of this chapter.

3.3 Microarchitecture of the selected parallel-templates design

This section describes the detailed microarchitecture of the selected parallel-templates architecture. It begins with a high-level overview of the accelerator organization, followed by a brief introduction of the LMEM and the TDL interfaces. The control datapath is then presented, as it generates the control signals required to coordinate the different functional blocks of the accelerator and supports the various operating modes. Finally, the core computational blocks, namely the correlation datapath, the signal energy computation, the RSQRT unit, and the NCC formation datapath are described in detail in the subsequent subsections.

3.3.1 Architecture overview

Figure 3.5 presents a block diagram of the complete implementation of the parallel-templates architecture. Input samples are streamed into the accelerator through the *data in* port and stored in the TDL, while templates are stored in the LMEM. The LMEM can be interfaced externally, allowing template updates through the *template in (tmpl in)* port.

A control datapath receives external control signals, most notably the *in valid* signal, which indicates the arrival of a new input sample. Based on these inputs, it generates internal control signals to coordinate the operation of the different functional blocks of the accelerator. Once the input sample has been successfully ingested, the control datapath asserts the *in ack* signal to acknowledge the transfer.

This handshake indicates that the *in valid* signal can be safely deasserted.

The correlation datapath and the signal energy datapath operate concurrently, both receiving data from the LMEM output (*tpl out*) and the TDL output (*tap out*). As per the parallel-templates principle, the correlation between the input signal window and the N_T templates is computed in parallel within the correlation datapath. While the TDL iterates over the samples of the input signal window, the signal energy datapath relies on control signals to select the relevant samples from *tap out* for the rolling-sum update.

At the end of a full sweep over the M samples stored in the TDL, the correlation coefficients are registered and made available on the *corr out* signal. A *corr out valid* signal, generated by the control datapath, indicates that the correlation results are valid and can be consumed. This event marks the transition between the first and second stages of the accelerator.

In the second stage, the RSQRT of the signal energy is computed to obtain the normalization factor. This factor is then multiplied sequentially with each correlation coefficient to produce the corresponding NCC coefficients. As they are computed, NCC values are registered and made available at the accelerator output. A *ncc out valid* signal is asserted for each NCC value to indicate its validity.

3.3.2 Input buffering and local memory

TDL Input samples received on the *data in* signal are buffered using a TDL. The TDL is implemented as a shift register composed of $M = 40$ registers of $W = 16$ bits each, with taps exposing all intermediate register outputs. A single output, *tap out*, provides access to the selected tap, as determined by the *tap select* signal. This selection signal has a width of $W_{\text{sel}} = \lceil \log_2(M) \rceil = 6$ bits. A *shift* control signal, generated by the control datapath, governs both the synchronous shifting of stored samples and the insertion of new input samples.

To avoid the need for explicitly resetting all 40 registers, an internal fill-level register is used. This $\lceil \log_2(M + 1) \rceil = 6$ -bit register tracks the number of valid samples currently stored in the TDL. It is incremented each time a shift occurs, up to a maximum value of M . During reset, only this fill-level register is cleared, while the data registers themselves are left unchanged. When the selected tap index exceeds the current fill level, the value presented at *tap out* is forced to zero, ensuring correct behavior during the initialization phase.

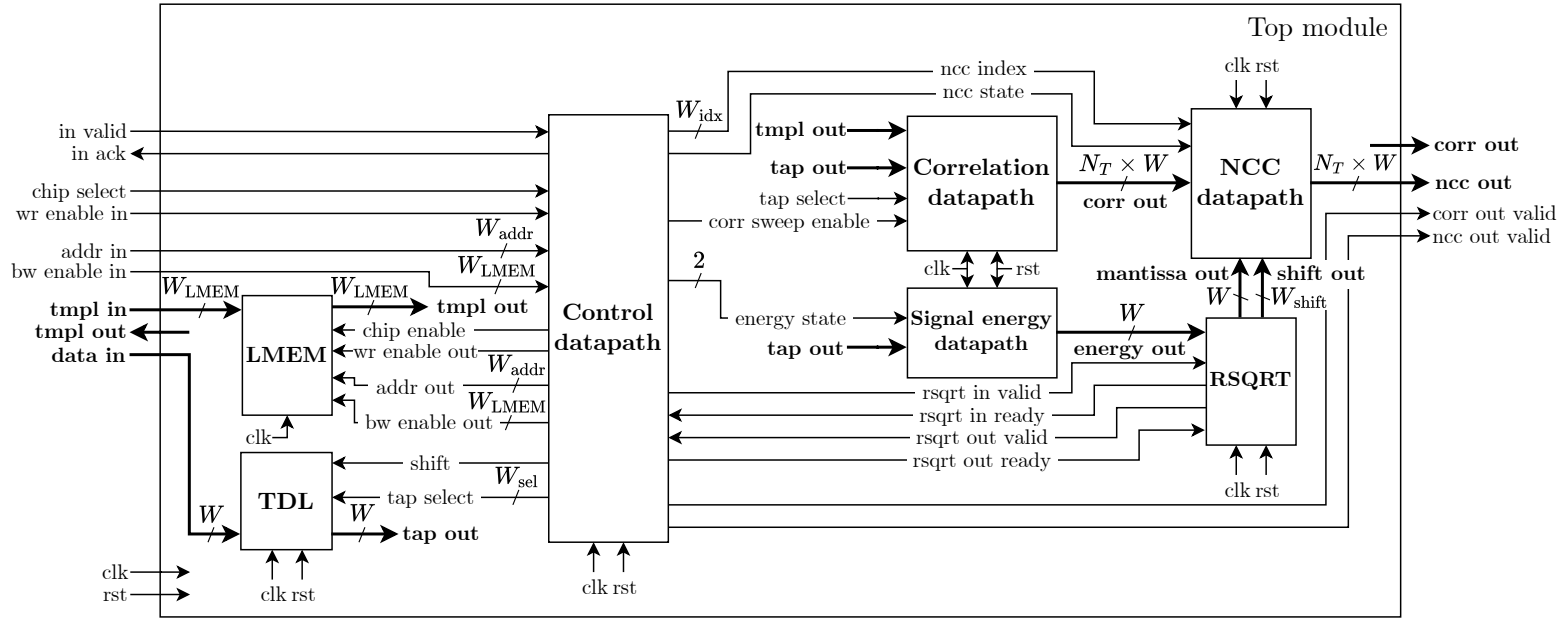


Figure 3.5: Block diagram of the microarchitecture of the parallel-templates design. Input samples are streamed into the TDL, while the LMEM stores the templates. A control datapath receives external control signals and generates internal control signals to coordinate the operation of the different functional blocks. The correlation datapath computes, in parallel, the correlations between the signal window and the templates. In parallel, the signal energy datapath computes the energy of the signal window. The resulting energy is forwarded to the RSQRT module, which produces the normalization factor. This factor is then used in the NCC datapath together with the correlation results to compute the NCC coefficients. The correlation and NCC outputs are made available at the accelerator interface along with their corresponding valid signals.

In steady-state operation, new input samples are provided at a rate of $f_s = 20$ kS/s, corresponding to one sample every f_{clk}/f_s clock cycles. The *tap select* signal sweeps sequentially from the first to the last tap, enabling the sequential access to all samples of the current signal window during correlation and energy computation.

LMEM Normalized templates are stored in the LMEM of the accelerator. As discussed in the first-order cost model, the LMEM is implemented using an SRAM macro optimized for the targeted technology. Templates are organized in a column-wise fashion: for each template, the first sample is stored in the first row and the last sample in the last row. This organization results in a required word width of $W_{\text{LMEM}} = N_T \times W = 128$ bits and a word depth of $M = 40$.

Access to the LMEM is controlled by the control datapath. When an external template update is requested via the *chip select* signal, the accelerator is stalled and the LMEM interface is made available externally. The *write enable* (*wr enable in*) signal indicates a write operation at the address provided by *addr in*, which has a width of $W_{\text{addr}} = \lceil \log_2(M) \rceil = 6$ bits. Fine-grain updates are supported through the *bit write enable* (*bw enable in*) signal. These control signals are routed through the control datapath and forwarded to the SRAM interface, together with a *chip enable* signal that allows clock-gating of the memory during cycles where no access is required.

The *tmpl in* port is used to write data into the LMEM, while the *tmpl out* port provides read access to template samples for the correlation datapath. In steady-state operation, template samples are read row by row in reverse order, at a rate of one word per clock cycle, to match the natural ordering of the input signal samples provided by the TDL. As the SRAM macro provides registered outputs, read accesses have a latency of one clock cycle. Consequently, the read address must be provided one clock cycle in advance so that valid data is available when required by the correlation datapath. Consequently, during idle cycles, e.g., when waiting for a new input sample, the address of the first word to be read is preloaded so that valid data is immediately available at the LMEM output at the start of the next computation.

3.3.3 Control datapath

In steady-state operation, a processing cycle is initiated when the *in valid* input signal is asserted, indicating the arrival of a new input sample. During the following clock cycle, the *in ack* signal is asserted by the accelerator, acknowledging that the

sample has been accepted and that the internal processing cycle can begin.

The control datapath orchestrates the operation of the first stage by sweeping over the taps of the TDL using the *tap select* signal and over the template samples using the LMEM address signal *addr out*. These two signals advance synchronously to ensure correct pairing between signal samples and template samples.

The correlation datapath is driven by two control signals. The *corr sweep enable* signal indicates the start of a new correlation sweep over the M samples. The *tap select* signal is used internally to control the accumulation process: at the first tap, accumulators are initialized using only the current product, while subsequent taps perform accumulation. At the final tap, the accumulated correlation results are registered and made available at the *corr out* outputs.

The signal energy datapath is controlled by a 2-bit state signal that specifies the operation to perform. The rolling-sum update is distributed across the sweep as follows: the addition of the newest squared sample is performed when the first tap is accessed, the subtraction of the previously stored oldest squared sample is performed on the second tap (although it could be scheduled at any other non-conflicting point), and the update of the stored squared sample corresponding to the current window is performed when the last tap is accessed. This scheduling ensures that all energy operations are completed within a single sweep.

At the completion of the sweep over the M taps, both the correlation results and the updated energy value are valid. The *corr out valid* signal is asserted to indicate that the correlation results can be consumed. Immediately afterwards, a new input sample can be ingested, allowing continuous operation without idle cycles.

A stalling mechanism is also implemented to support template updates during operation. When a template update is requested, the control datapath temporarily halts the progression of the sweep. During this pause, the correlation datapath recomputes results using the updated template samples, while the signal energy datapath resumes its operations once the stall is released.¹

The second stage begins once the energy value becomes available, which occurs after completion of the subtraction operation in the rolling-sum update. At this point, the RSQRT module is triggered to compute the normalization factor. The

¹In the current implementation, this stalling mechanism exhibits a limitation: if a stall occurs during a sweep, the companion register is not updated. In addition, if a stall occurs between the addition and subtraction steps of the rolling-sum update, the subtraction operation may be skipped. As a result, the energy computation becomes inconsistent, and template updates during operation are therefore disabled in the present design.

interface with the RSQRT module follows a standard handshake protocol: *in valid* indicates that a new input value is available, *in ready* indicates that the module can accept it, *out valid* signals that the result is available, and *out ready* indicates that it has been consumed.

Once both the normalization factor and the correlation coefficients are available, the NCC datapath is activated. The *ncc state* signal indicates when NCC computation can begin. The *ncc index* signal selects the appropriate correlation coefficient from *corr out* and determines the destination register for the corresponding NCC value. Each NCC coefficient is produced by multiplying the normalization factor with the selected correlation value. As each result is computed, it is registered, and the *ncc out valid* signal is asserted to indicate its availability.

3.3.4 Correlation datapath

The correlation datapath consists of N_T MAC lanes, each associated with one template. Each lane receives a W -bit subset of the *tmpl out* signal, corresponding to one template sample, and the current signal sample provided by *tap out*. Multiplication is performed combinationally and preserves full precision, producing a 32-bit result. This product is then sign-extended to match the accumulator width, i.e., $2W + \lceil \log_2(M) \rceil = 38$ bits, and accumulated in the corresponding register. After completion of the accumulation, the correlation results are reduced in width and saturated to a Q15 fixed-point format before being registered and made available on the *corr out* outputs.

One MAC operation is performed per cycle in each lane, and after M cycles, all correlation values are available. At the start of a sweep, the accumulator registers are initialized with the first product rather than accumulating with a previous value. Upon reset, both the accumulator registers and the output registers are cleared to zero.

3.3.5 Signal energy datapath

The signal energy datapath implements a rolling sum of squared input samples provided by the *tap out* signal. Each signal sample is first squared, producing a 32-bit result. This value is then sign-extended to match the accumulator width, i.e., 38 bits, before being accumulated. The resulting energy is finally reduced in width and saturated to a Q15 fixed-point format before being made available on the *energy out* signal.

As described previously, the rolling sum is implemented using a companion register that stores the squared value of the oldest sample in the current signal window. The update equation of the rolling sum is given in (3.7), where the first term corresponds to the previously accumulated energy, the second term corresponds to the squared value of the newest sample entering the window, and the third term corresponds to the squared value of the oldest sample leaving the window, stored in the companion register:

$$E[\tau + 1] = E[\tau] + s^2[\tau + M] - s^2[\tau] \quad (3.7)$$

From a timing perspective, the first two clock cycles of the update perform the addition of the newest squared sample and the subtraction of the oldest squared sample, respectively. A third clock cycle is used to reduce the accumulator width and apply saturation, after which the energy value is made available to the RSQRT module. Upon reset, the energy accumulator, the companion register, and the output register are all cleared to zero. During the first $M - 1$ processing cycles, the energy value is computed as if the input signal were zero-padded, since the TDL is not yet fully populated.

3.3.6 RSQRT datapath

The RSQRT module is implemented as a three-stage pipeline. The first stage captures the *energy out* signal from the signal energy datapath. The second and third stages perform the reciprocal square root computation.

In the second stage, the input Q15 signal is first normalized. The sign of the input is checked. If a negative value is detected, the output is forced to zero. The position of the most significant bit (MSB) is then identified, and the input is left-shifted such that the MSB is aligned to bit position $W - 2$. This operation normalizes the input to the range $[0.5, 1)$. The four most significant fractional bits of the normalized value are then used to index a lookup table (LUT), providing an initial estimate of the RSQRT.

The LUT contains 16 entries corresponding to uniformly spaced values in the normalization range, defined as $x_i = 0.5 + i/32$. Each entry stores the RSQRT value in Q1.14 fixed-point format. This coarse approximation yields a mean absolute error of 2.12% and a maximum absolute error of 51.19%.

To improve accuracy, a single Newton iteration is performed on the LUT

estimate. The update equation is given by

$$y_{n+1} = y_n \left(1.5 - \frac{x_{\text{norm}} y_n^2}{2} \right), \quad (3.8)$$

where x_{norm} is the normalized input value and y_n is the initial RSQRT estimate from the LUT, both represented in **Q1.14** format. The division by 2 is implemented as a right shift. This refinement reduces the mean absolute error to 0.05%, with a maximum error of 1.77%.

The resulting RSQRT value is then denormalized. To preserve precision, the result is represented using two signals: a *mantissa out*, formatted as a **Q15** value, and a *shift out* signal indicating the number of left shifts required to reconstruct the final result.

Overall, the RSQRT computation requires three clock cycles ($N_{\text{SQ}} = 3$). Taking into account the latency of the preceding stages, a total of six clock cycles elapse between the ingestion of a new sample and the availability of the corresponding normalization factor.

3.3.7 NCC formation datapath

The NCC formation datapath combines the normalization factor with the correlation results to produce the final NCC coefficients. For each of the N_T correlation values, a single multiplier is reused sequentially. At each cycle, one **Q15** correlation coefficient from the *corr out* signal is selected and multiplied by the normalization factor provided on the *mantissa out* signal.

The resulting 32-bit product is then left-shifted according to the value of *shift out*, which encodes the denormalization factor. The shifted value is subsequently reduced in width and saturated to the **Q15** format. Each NCC coefficient is registered and made available at the accelerator output through the $(N_T \times W)$ -bit *ncc out* signal.

The NCC computation is performed sequentially over N_T cycles and can be overlapped with the computation of the next set of correlation coefficients, ensuring that the NCC stage does not limit the overall throughput. Unlike the *corr out* outputs, the *ncc out* registers are not reset to zero.

3.3.8 Implementation summary

The implemented parallel-templates architecture achieves a steady-state throughput of one processed input sample every $M = 40$ clock cycles, consistent with the CPS values derived in the first-order cost model. This ensures compatibility with the target input rate at sub-MHz clock frequencies.

From a latency perspective, M cycles are required to complete the correlation accumulation for a given input sample. Overlapping with this, a total of $3 + N_{\text{SQ}} = 6$ cycles are required to obtain the normalization factor. An additional $N_T = 8$ cycles are needed for the sequential NCC formation. As these operations are pipelined, the end-to-end latency is dominated by the correlation stage and can be translated as $M + N_T = 48$ cycles. This confirms that the correlation datapath primarily dictates the overall latency.

The two-stage operation of the accelerator is enabled by the scheduling of computations. The RSQRT and NCC operations are completed within the M -cycle budget associated with the correlation accumulation of the next input sample. Consequently, the second stage does not stall the first stage, and continuous processing is achieved without additional buffering or idle cycles. The control datapath ensures that all data dependencies are satisfied while maintaining pipeline efficiency.

Overall, the implementation validates the assumptions made in the first-order model and achieves a balanced trade-off between performance and hardware cost. The parallel computation of correlations, combined with the sequential reuse of arithmetic resources in the second stage, enables real-time operation at low clock frequencies while limiting design complexity.

3.4 Verification approach and implementation toolflow

To verify the functionality of the accelerator and evaluate its numerical precision, the following methodology is adopted. A behavioral simulation is performed using, as input stimulus, one of the 30-minute preprocessed VENG recordings and a subset of eight templates selected from the twelve-template bank. This subset corresponds to the most frequently selected templates by the trace aggregation stage in the reference software implementation.

In software, four implementations of the NCC computation are evaluated: a

floating-point full-precision version, the previously presented reference implementation, the reduced-precision implementation, and a further reduced-precision version using Q7 arithmetic.

The floating-point implementation serves as the gold standard. Absolute errors for both the correlation results and the NCC coefficients are computed for each of the other software implementations, as well as for the hardware implementation, with respect to this reference. Statistical metrics of these errors are then evaluated and compared to assess the numerical accuracy of the proposed design.

From a technical standpoint, the implementation of the accelerator and its associated testbenches is carried out using the SystemVerilog hardware description and verification language. Behavioral simulations are performed using QuestaSim from Siemens EDA. Logic synthesis is conducted using the Design Compiler tool from Synopsys, targeting the TSMC 65 nm technology node.

Post-synthesis structural simulations are also performed using QuestaSim, from which switching activity annotations are extracted. These annotations are then used to perform a second logic synthesis run, enabling more accurate power estimation. Physical synthesis is not performed in this work. Consequently, the reported PPA metrics are derived solely from logic synthesis.

Chapter 4

Characterization and discussion

This chapter presents the characterization of both the software and hardware components developed in this work. The evaluation and measurement methodology is first detailed, followed by the presentation and analysis of the obtained results. Finally, the integration perspective of the proposed solution within the BREWER platform [8] is explored, and a comparison with existing neural spike processing systems is provided.

4.1 Software implementation characterization

In this section, the different software implementations introduced in Chapter 2, namely the 32-bit reference implementation, the 16-bit reduced-precision implementation, and the HW/SW-compatible implementation, are characterized. First, their functional performance is evaluated using the metric score (MSC) defined in Chapter 1. Then, their execution time, power consumption, and memory utilization are analyzed on the target evaluation platform.

4.1.1 Evaluation and measurement setup

As introduced in Section 1.3.3, a methodology is used to evaluate the ability of the algorithm to detect seizure events and to compare different implementations. The metric score (MSC) is defined as the ratio between the RMS value of the metric trace $m[m]$ over the seizure interval and over the baseline interval. Higher MSC values indicate better discrimination between these two regimes. In the provided dataset, six recordings, denoted P1 to P6, are processed using each algorithm

implementation along with their corresponding preprocessing steps. For each recording and each implementation, an MSC value is computed. An average MSC across all recordings is also reported.

Execution time, power consumption, and memory utilization are evaluated on a NUCLEO-L4A6ZG development board from STMicroelectronics. This board integrates an STM32L4A6ZG microcontroller featuring a Cortex-M4 processor, 320 kB of RAM, and clock frequencies up to 80 MHz. Unless otherwise specified, a processor clock frequency of 48 MHz is used, in line with [6]. Among the available power modes of the MCU, two are considered. The run mode, operating with the main voltage regulator in Range 1, is required to support the selected clock frequency. The idle mode corresponds to the Stop 2 mode, in which the low-power voltage regulator remains active while the processor, flash memory, and peripherals are powered down.

Processing is triggered by interrupts generated via a user button, allowing the MCU to wake up from the idle state and execute the processing sequence. Execution time is measured using the built-in data watchpoint and trace cycle counter, which provides cycle-accurate measurements without perturbing execution. Cycle counts are obtained for each task independently and averaged over 100 iterations. Execution times are then derived using the processor clock frequency.

Power consumption is measured using a Model 2450 source measure unit (SMU) from Keithley. A supply voltage of 1.8 V is provided, and high time-resolution current measurements are recorded to capture detailed power traces. From these measurements, power consumption is extracted for both run and idle modes.

Finally, memory utilization is obtained after compilation of the code base. A detailed breakdown is derived from the generated `.map` file.

4.1.2 Functional evaluation

The functional performance of the algorithm is evaluated using the methodology described previously, and the results are reported in Table 4.1. The baseline implementation presented in [6], when applied to the dataset used in this work, achieves an average MSC of 3.45. This value shows limited variation across recordings, ranging from 4.80 for P1 to 1.66 for P6.

Among the three implementations considered, the reference implementation achieves the highest MSC values, with an average of 11.48 and a peak value of 26.55 for P1. The reduced-precision implementation yields intermediate results, outperforming the baseline on most recordings.

	Metric score (MSC)						
	P1	P2	P3	P4	P5	P6	Avg.
Baseline	4.80	3.50	2.55	3.93	4.28	1.66	3.45
Reference	26.55	1.73	7.74	12.50	4.49	15.84	11.48
Reduced-precision	23.30	2.61	9.58	6.01	4.05	11.38	9.49

Table 4.1: Metric scores obtained for each implementation across six recordings (P1–P6). For each recording, the highest score is highlighted in bold. The full-precision reference implementation achieves the highest average MSC, while the reduced-precision implementation generally outperforms the baseline approach.

An exception is observed for P2, where the baseline implementation achieves the highest MSC. Notably, all implementations yield relatively low MSC values for this recording, suggesting that the selected templates may be poorly matched to this dataset or that other limiting factors affect the algorithm performance for this specific case.

Overall, the increase in MSC observed for both the reference and reduced-precision implementations indicates that the optimizations introduced in this work improve the robustness of the algorithm. Although the reduced-precision implementation exhibits slightly lower MSC values than the reference, it provides a favorable trade-off between numerical precision and implementation cost. This result demonstrates that reduced-precision computation can preserve most of the algorithm’s discriminative capability while significantly lowering implementation complexity.

4.1.3 Execution time characterization and scalability

A detailed breakdown of the execution time for each task of the baseline, reference, and reduced-precision implementations is provided in Figure 4.1. Baseline results are directly taken from [6], while the execution times of the reference and reduced-precision implementations are measured as described in Section 4.1.2. To enable a fair comparison, a single template is used for this detailed breakdown.

It should be noted that only the execution time of the NCC computation depends on the number of templates, whereas the peak finding, AP validation, and metric computation tasks remain independent of this parameter. Similarly, in the baseline implementation, the filtering, clustering, and metric detection tasks exhibit constant execution time regardless of the number of templates.

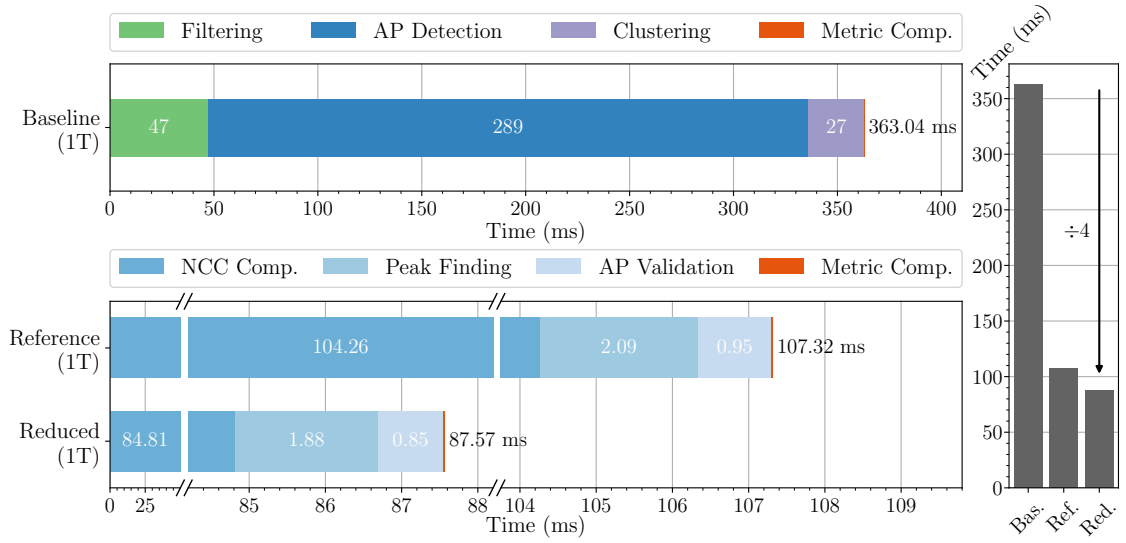


Figure 4.1: Execution-time breakdown across tasks for the three implementations, shown as a bar chart. For each implementation, measurements are obtained at a clock frequency of 48 MHz using a single template. The total execution time is indicated at the end of each bar. In the baseline implementation, the AP detection task corresponds to the combined execution time of the NCC computation, peak detection, and AP validation stages. A fourfold reduction in execution time is observed between the baseline and the reduced-precision implementation.

As shown in the bar charts, a significant reduction in execution time is observed between the baseline implementation and the reference and reduced-precision implementations. This improvement is partly due to the removal of the filtering task, which is handled as a preprocessing step in this work. However, the main contribution comes from the AP detection stage, and in particular from the NCC computation. A complete rewrite of this task enables a reduction in execution time by a factor of 2.69 for the reference implementation and 3.3 for the reduced-precision implementation. Overall, this work provides a more efficient computation of NCC coefficients.

Consequently, a fourfold reduction in total execution time is achieved between the baseline and the reduced-precision implementations. In terms of duty cycle, the active time decreases from 72.61 % for the baseline to 17.51 % for the reduced-precision implementation. This reduction allows the MCU to spend a larger fraction of time in low-power modes, resulting in significant power savings.

Figure 4.2 presents the execution time breakdown for the HW/SW-compatible implementation. This implementation is tightly coupled to the hardware accelerator

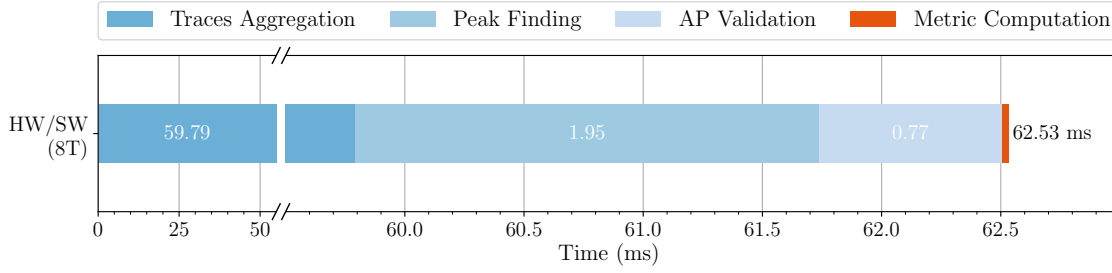


Figure 4.2: Execution-time breakdown for the HW/SW-compatible implementation, shown as a bar chart. For this implementation, the number of templates is $N_T = 8$. An even lower execution time is achieved compared to the reduced-precision implementation despite processing eight templates, highlighting the efficiency of having a hardware accelerator.

parameters, in particular the number of templates ($N_T = 8$). In this configuration, the NCC computation task is offloaded to hardware and is therefore removed from the software execution path. The remaining real-time task on the MCU is the traces aggregation, which operates following the data acquisition schedule described in Chapter 2.

The traces aggregation task is executed every 50 μ s. The reported execution time corresponds to the cumulative execution time over 10 000 iterations, which are required to fill the buffer used by the main processing stage. Once the buffer is filled, the main processing stage executes the peak finding, AP validation, and metric computation tasks.

From a real-time perspective, the traces aggregation task satisfies the first hard real-time constraint. A single execution requires 287 clock cycles, corresponding to 5.98 μ s at 48 MHz. Within the 50 μ s period, this results in an active duty cycle of 11.9 %. After buffer completion, the main processing stage executes in 2.74 ms. This value excludes the 61 interruptions caused by trace aggregation during its execution.

Overall, the second real-time constraint, requiring completion of the remaining algorithm pipeline within 500 ms, is comfortably satisfied. The resulting effective duty cycle, including trace aggregation during main processing, is 12.51 %, representing a reduction of 5 percentage points compared to the reduced-precision software implementation, while enabling the use of eight templates. This demonstrates that the proposed HW/SW-compatible implementation not only meets real-time constraints but also supports scalable operation with reduced processor utilization, enabled by the use of a hardware accelerator.

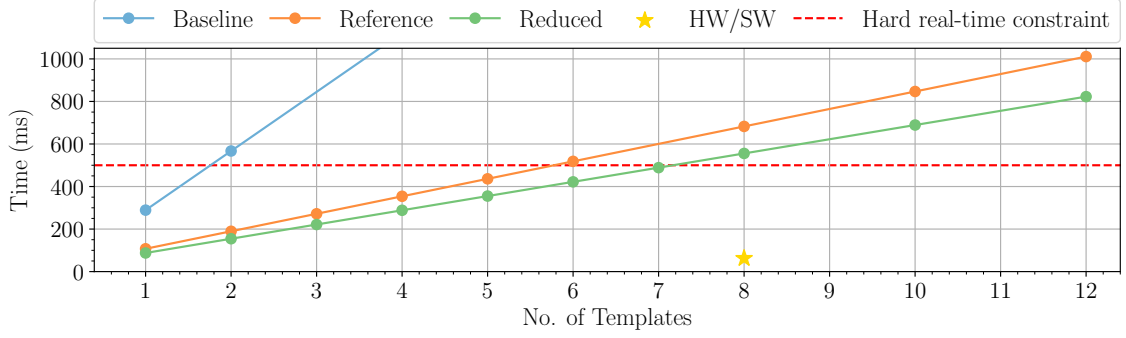


Figure 4.3: Total execution time as a function of the number of processed templates for the three software implementations and the HW/SW-compatible implementation. The baseline implementation violates the real-time constraint as soon as more than one template is used, while the reference and reduced-precision implementations support up to six and seven templates, respectively. The HW/SW-compatible implementation, highlighted by a yellow star, processes eight templates with an execution time of 62.53 ms. Overall, the use of a hardware accelerator enables the processing of a higher number of templates while maintaining a low and nearly constant execution time.

Figure 4.3 provides insight into the scalability of the total execution time of the presented software implementations with respect to the number of templates. Constrained by the main real-time requirement of processing 500 ms of signal within a 500 ms time frame, the baseline implementation from [6] exceeds this limit as soon as more than one template is used. In this work, this limit is extended to six and seven templates for the reference and reduced-precision implementations, respectively. This improvement results from a combination of lower constant execution time and a reduced growth rate of execution time as the number of templates increases. The reduced-precision implementation exhibits an even smaller slope than the reference implementation, which can be attributed to the use of single instruction, multiple data (SIMD) operations for 16-bit arithmetic, in contrast to 32-bit operations.

The HW/SW-compatible implementation is represented by a yellow star at the eight-template point. As expected, this implementation outperforms the software-only approaches thanks to hardware acceleration. While the number of templates is fixed in this configuration, as it is constrained by the hardware accelerator design, the execution time is largely decoupled from the number of templates. Consequently, a low execution time can be maintained even as the number of templates increases, provided that the accelerator is dimensioned accordingly.

Overall, the execution-time analysis demonstrates substantial improvements over the baseline implementation from [6]. Furthermore, the HW/SW approach is strongly motivated by the significantly reduced processor duty cycle, enabling both improved scalability and enhanced energy efficiency.

4.1.4 Power characterization

Power measurements and corresponding duty cycles for the reference and reduced-precision implementations are reported in Table 4.2. The power consumption in both run and idle modes is provided.

The baseline power consumption is not directly taken from [6], as a different evaluation platform was used. Instead, a projection is performed by combining the duty cycle reported in [6] with the power measured for the reference implementation. The idle power consumption is assumed to be identical across all implementations.

A reduction of 6.35 mW is observed between the projected baseline and the reference implementation. This decrease is primarily attributed to the significant reduction in duty cycle, which allows the MCU to spend more time in low-power idle mode. The reduced-precision implementation exhibits a slightly higher run-mode power consumption, which can be attributed to the use of SIMD instructions that increase instantaneous processor activity despite improving computational efficiency.

Overall, the reduced-precision implementation achieves a total power consumption of 2.45 mW, representing a substantial improvement over the projected baseline while maintaining high computational performance.

Table 4.3 reports the power consumption of the different operating modes of the HW/SW-compatible implementation, along with the associated duty cycles, for three processor clock frequencies. The "aggr." label refers to the trace aggregation task, while "main" corresponds to the remaining processing pipeline. At 48 MHz, the run mode corresponds to the configuration described in Section 4.1.2. For lower clock frequencies (24 MHz and 8 MHz), the main voltage regulator is switched to Range 2 to reduce power consumption.

At a given clock frequency, the aggregation and main tasks exhibit similar power consumption, indicating a comparable level of processor activity in both cases. Overall, reducing the clock frequency does not significantly decrease total power consumption, as the increase in duty cycle compensates for the reduction in instantaneous active power.

Implementation	Duty cycle (%)	Power consumption (mW)		
		Run	Idle	Total
Projected Baseline	72.61	12.62	0.22	9.23
Reference	21.46	12.62	–	2.88
Reduced-precision	17.51	12.94	–	2.45

Table 4.2: Duty cycle and power consumption measurements for the three software implementations using a single template. The duty cycle represents the proportion of time spent in the run mode. The baseline implementation is projected using the run-mode power measured for the reference implementation and assuming identical idle power across implementations. The reduced-precision implementation achieves a significant reduction in total power consumption, with a factor of 3.77 compared to the projected baseline.

f_{CPU} (MHz)	Duty cycle (%)		Power consumption (mW)			
	Aggr.	Main	Aggr.	Main	Idle	Total
48	11.96	0.55	11.65	12.91	0.22	1.66
24	23.92	1.10	5.93	6.23	–	1.65
8	71.75	3.29	2.14	2.35	–	1.66

Table 4.3: Duty cycle and power consumption of the HW/SW-compatible implementation for different CPU clock frequencies. The duty cycle and power consumption are reported separately for the aggregation task (Aggr.) and the main processing task. While the duty cycle varies significantly with clock frequency, the total power consumption remains nearly constant, indicating a balance between execution time and dynamic power.

The lowest total power consumption is observed at 24 MHz, reaching 1.65 mW. This represents a reduction of approximately 800 μW compared to the reduced-precision software implementation, while supporting an increase in the number of templates from one to eight. Although the power consumption of the hardware accelerator itself is not included in this measurement, this result constitutes a significant step toward an ultra-low-power implementation of the seizure detection algorithm proposed in this work.

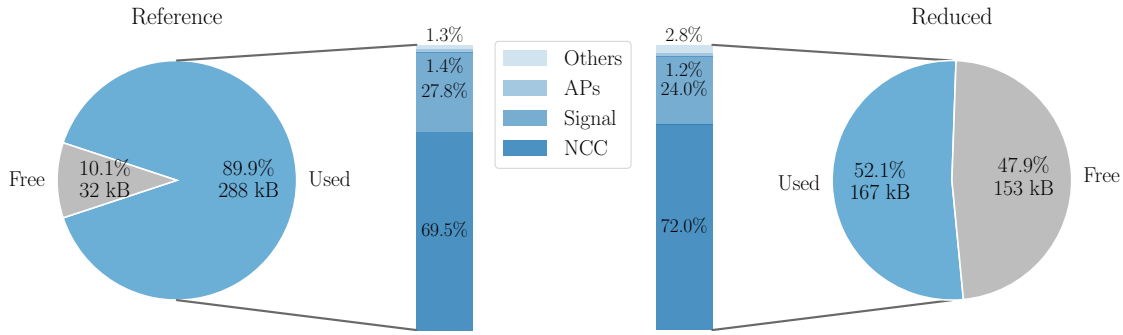


Figure 4.4: Memory utilization for the reference and reduced-precision implementations, shown as pie charts, with detailed breakdowns provided in adjacent bar charts. The reference implementation, based on 32-bit data, requires larger memory buffers compared to the reduced-precision 16-bit implementation highlighting the impact of data width reduction on overall memory requirements. In both cases, the data structures associated with the NCC computation dominate memory usage, followed by the signal double buffer.

4.1.5 Memory utilization

The 320 kB RAM utilization for the reference and reduced-precision implementations is reported in Figure 4.4, along with a breakdown of memory usage across the different tasks.

Transitioning from a 32-bit to a 16-bit implementation results in a reduction of 121 kB, corresponding to 38 % of the total available memory. The majority of memory usage is associated with the NCC computation task, followed by signal buffering and the data structures used to store AP features.

The memory utilization of the HW/SW-compatible implementation, shown in Figure 4.5, further highlights the benefits of hardware acceleration. A reduction of 79 kB is achieved compared to the reduced-precision implementation, primarily due to the removal of input signal buffering and the working buffers required for NCC computation.

With such a reduced memory footprint, it becomes feasible to relocate data previously stored in flash memory, such as program instructions, into RAM. This enables the potential deactivation of the flash memory during operation, resulting in additional power savings. Furthermore, the target platform allows for the deactivation of 64 kB of RAM, which could be leveraged to further reduce overall power consumption.

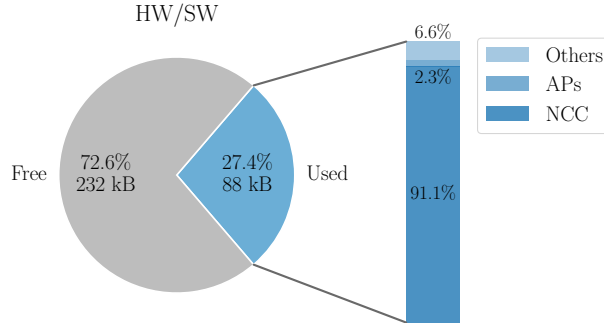


Figure 4.5: Memory utilization for the HW/SW-compatible implementation, with a detailed breakdown. This implementation exhibits even lower memory usage than the reduced-precision implementation. This improvement stems from the offloading of the NCC computation to hardware, which eliminates the need for additional working buffers in software beyond those required to store the aggregated traces.

4.2 Hardware accelerator

In this section, the hardware accelerator designed and implemented in Chapter 3 is characterized. First, the numerical accuracy of the correlation and NCC outputs is evaluated against software-based implementations with different precision levels. Next, post-synthesis power and area metrics are reported. Finally, an analysis of timing closure, latency, and throughput is presented.

4.2.1 Evaluation and synthesis setup

The methodology used to evaluate the functionality of the hardware accelerator is detailed in Section 3.4. A behavioral simulation is performed using recording P1, which contains approximately 41 million samples, along with eight templates. The results are compared against three software-based implementations: a 32-bit version equivalent to the reference implementation, a 16-bit version corresponding to the reduced-precision implementation, and a further reduced-precision version using 8-bit arithmetic. These implementations are realized using the CMSIS-DSP Python wrapper.

A floating-point computation of the correlation and NCC coefficients is used as the ground truth. Absolute errors are computed with respect to this reference, and the following statistical metrics are reported: the mean, as well as the 1st, 25th, 50th, 75th, and 99th percentiles.

Power and area results are derived from logic synthesis. A clock period of 1000 ns is assumed, corresponding to a clock frequency of 1 MHz. The clock uncertainty is set to 0.5 ns, the output pin load to 0.01 pF, and the input transition time to 0.1 ns. Switching activity annotations obtained from post-synthesis structural simulations are used to perform a second synthesis run, enabling a more accurate estimation of power consumption.

4.2.2 Numerical validation

Figure 4.6 presents box plots of the absolute error distributions for the correlation and NCC outputs across the three software implementations and the hardware accelerator.

For the correlation results, the absolute error distribution of the hardware accelerator matches that of the Q15 software implementation. Both distributions, together with the Q7 implementation, are centered around 7.48×10^{-3} , whereas the Q31 implementation exhibits significantly lower errors, with a median of approximately 1.5×10^{-6} . The spread of the distributions for the Q31, Q15, and hardware implementations is comparable, while the Q7 implementation shows a much wider distribution, with a 99th percentile reaching up to 1.16. The mean absolute error of the Q7 implementation is also significantly higher, at 5.34×10^{-2} , compared to 1.41×10^{-3} for both the Q15 and hardware implementations.

For the normalized cross-correlation results, the differences between implementations are more pronounced. The Q31 implementation again achieves the lowest errors, with a median of 6.28×10^{-5} and a 99th percentile below 7.79×10^{-4} . The Q15 implementation exhibits a median of 1.94×10^{-3} and a 99th percentile of 2.49×10^{-2} . In contrast, the hardware accelerator achieves improved accuracy compared to the Q15 implementation, with a median of 2.36×10^{-4} and a 99th percentile of 8.94×10^{-4} . This places the hardware accelerator performance between the Q31 and Q15 software implementations.

These results highlight the benefit of maintaining higher internal numerical precision within the hardware datapath for accumulation and multiplication operations, despite using 16-bit interfaces. As expected, the Q7 implementation exhibits the highest absolute errors. Nevertheless, these results do not preclude the possibility of designing a reliable 8-bit-based seizure detection system, especially when combined with hardware acceleration techniques that help preserve numerical accuracy.

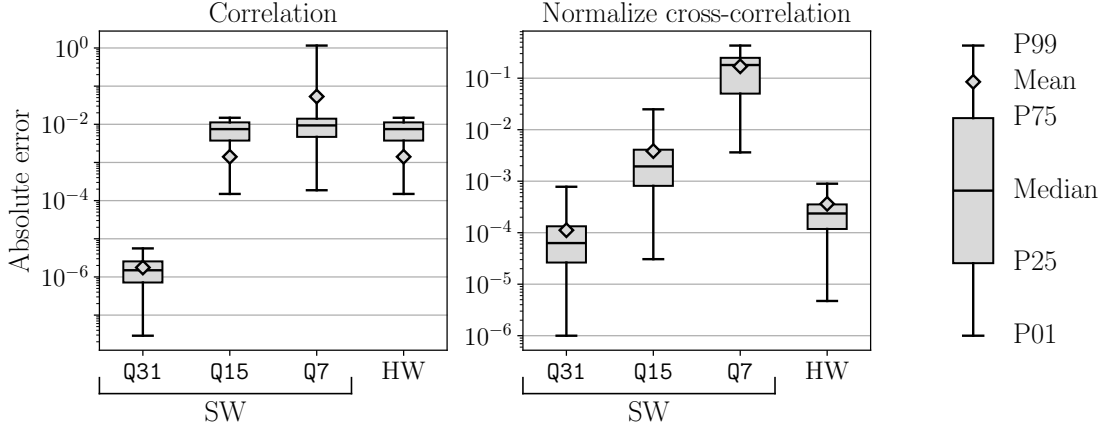


Figure 4.6: Box plots of the absolute error distributions for the correlation and NCC signals across three software implementations and the implemented hardware accelerator. The software implementations range from 32-bit to 8-bit precision, while the hardware accelerator interfaces using 16-bit values but employs wider internal datapaths to improve accuracy. For correlation, the absolute error distribution of the hardware accelerator matches that of the Q15 implementation, with a median of approximately 7.48×10^{-3} . For the NCC results, the hardware accelerator achieves an intermediate accuracy between the Q31 and Q15 implementations, with a limited spread of errors and a median of 2.36×10^{-4} .

4.2.3 Post-synthesis power and area

Figure 4.7 reports the distribution of power consumption and cell area across the different modules of the hardware accelerator. The total power consumption obtained after the second logic synthesis pass is 53.48 μ W. Although this value does not reflect the consumption of a fully fabricated design, it provides a valid order-of-magnitude estimate. Compared to the power consumed by the MCU, this contribution remains marginal.

The total power is distributed across the various modules of the accelerator. In particular, the LMEM implemented using register-file-based SRAM exhibits a power consumption of 10.2 μ W. Alternative memory implementations were also evaluated: logic-based SRAM and SRAM macros exhibit higher power consumption, at approximately 53.5 μ W and 39.4 μ W, respectively. Overall, memory modules, including the TDL and the LMEM, account for a significant fraction of the total power consumption.

The total cell area reported is 0.062 mm^2 , with the LMEM accounting for more than one quarter of this area. Once again, the register-file-based SRAM provides

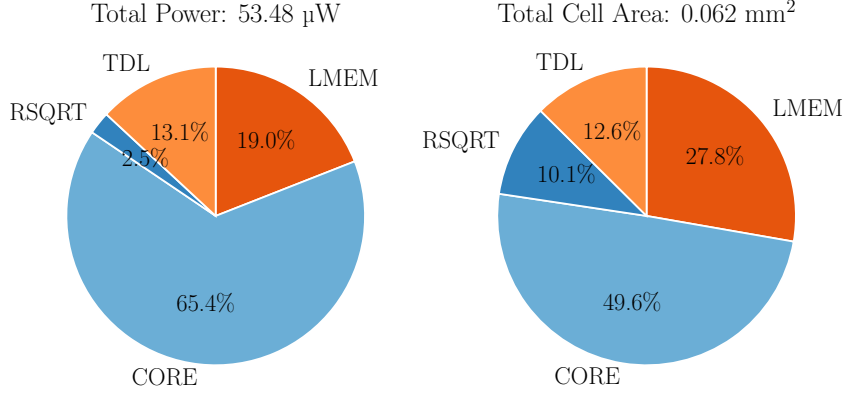


Figure 4.7: Pie charts showing the hierarchical distribution of power and cell area obtained from logic synthesis. The total power consumption of 53.48 μW is negligible compared to the 1.66 mW consumption of the HW/SW-compatible software implementation. The core accelerator dominates both power consumption and cell area. The use of register-file-based SRAM macros enables low memory power while maintaining a compact area footprint.

the most area-efficient solution, occupying 0.017 mm^2 , compared to 0.068 mm^2 for logic-based SRAM and 0.033 mm^2 for SRAM macros. Overall, memory structures represent nearly half of the total area when including the LUT used in the RSQRT module. It should be noted that these estimates correspond to post-synthesis cell area and do not include routing overhead. With the exception of the SRAM-based LMEM, the remaining logic area is expected to increase by at least a factor of two after physical synthesis.

4.2.4 Timing closure, latency and throughput

At a clock frequency of 1 MHz, a worst-case slack of 983.87 ns is reported, with the critical path located within the RSQRT module. This result highlights the significant timing margin of the accelerator and motivates the exploration of low-power techniques that can exploit this margin to further reduce power consumption.

As a reminder, the accelerator achieves a steady-state throughput of one processed input sample every $M = 40$ clock cycles, with an overall latency of 48 cycles. Based on these figures, the clock frequency can be reduced to 800 kHz while still satisfying the real-time constraint imposed by the 20 kS/s sampling rate.

4.3 Integration perspective and system comparison

This section positions the proposed hardware accelerator within a realistic system context by first evaluating its integration within the BREWER platform introduced in [8]. The resulting system-level performance is then compared with existing neural spike processing systems.

4.3.1 Projection on the BREWER platform

To assess the system-level relevance of the proposed hardware accelerator, a projected integration within the BREWER platform introduced in [8] is considered. The BREWER chip integrates an ultra-low-power dual-channel AFE tailored for VENG acquisition in an implantable system. In addition, it includes an on-chip DBE implementing an IIR decimation and filtering stage, and embeds a Cortex-M4 MCU.

In the projected integration considered here, the proposed accelerator is placed between the DBE and the embedded Cortex-M4. Acquisition, filtering/decimation, and correlation/NCC computation are therefore performed in hardware, while the remaining stages of the detection pipeline are executed in software on the embedded MCU. Based on this partitioning, a system-level power consumption projection is derived in the following.

AFE and DBE In [8], the AFE+DBE supply current is reported as 9.8 μA at 1.2 V, which corresponds to a power consumption of

$$P_{\text{AFE+DBE}} = 9.8 \mu\text{A} \times 1.2 \text{ V} = 11.76 \mu\text{W}. \quad (4.1)$$

Post-layout evaluation further indicates that the DBE accounts for approximately 1.13 μW of this budget [8].

MCU/software projection As no Cortex-M4 power figures are reported in [8], the MCU energy is estimated using results from [18]. In that work, a software execution of a 4096-point FFT on a Cortex-M4 synthesized in the same 65 nm technology reports an energy of $E_{\text{FFT}} = 23354.7 \text{ nJ}$ at $f_{\text{clk}} = 128 \text{ MHz}$ and $V = 1.2 \text{ V}$, with an execution time of $t_{\text{FFT}} = 2.537 \text{ ms}$. From these values, a

first-order normalized dynamic current can be derived as

$$I_{\text{dyn,norm}} = \frac{E_{\text{FFT}}}{V t_{\text{FFT}} f_{\text{clk}}} = 59.93 \text{ } \mu\text{A/MHz}, \quad (4.2)$$

which is used here as a proxy for the Cortex-M4 dynamic consumption in run mode.

Using the HW/SW-compatible implementation operating point ($f_{\text{clk}} = 24 \text{ MHz}$), the run-mode dynamic power is estimated as

$$P_{\text{dyn,run}} \approx V \cdot I_{\text{dyn,norm}} \cdot f_{\text{clk}} = 1.2 \text{ V} \times 59.93 \text{ } \mu\text{A/MHz} \times 24 \text{ MHz} = 1.73 \text{ mW}. \quad (4.3)$$

With an effective software duty cycle of 25%, the average dynamic contribution becomes $0.25 \times 1.73 \text{ mW} = 432.5 \text{ } \mu\text{W}$. A static component is added to account for circuitry remaining powered during idle periods. Using the Stop 2-mode consumption reported for the STM32L4A6xG MCU family [19], the static power is estimated at $4.63 \text{ } \mu\text{W}$. The resulting software-related power estimate is therefore

$$P_{\text{SW}} = 432.5 \text{ } \mu\text{W} + 4.63 \text{ } \mu\text{W} = 437.13 \text{ } \mu\text{W}. \quad (4.4)$$

The additional cost of DMA transactions required to fetch accelerator outputs while the MCU is in low-power mode is not included in this estimate.

Hardware accelerator The accelerator power reported in this work ($P_{\text{HW}} = 53.48 \text{ } \mu\text{W}$) is obtained from post-logic-synthesis power estimation with activity annotation, and physical synthesis is not performed. Consequently, this value should be interpreted as an order-of-magnitude estimate. A post-layout implementation would typically increase the power due to routing and parasitic effects. Therefore, a conservative margin of +20% to +50% is expected.

Total power budget Using the above estimates, the projected total power consumption is

$$P_{\text{total}} = P_{\text{AFE+DBE}} + P_{\text{SW}} + P_{\text{HW}} = 502.37 \text{ } \mu\text{W}. \quad (4.5)$$

Compared to the 1.92 mW reported in [6], this corresponds to an overall reduction by a factor of approximately 3.8. In this projected budget, software processing remains the dominant contribution, indicating that further system-level gains would primarily come from reducing MCU activity and from tighter integration of the accelerator data path.

4.3.2 Comparison with neural spike processing systems

To position the proposed seizure detection system with respect to existing work, two recent neural spike processing systems are considered. [20] and [21] present low-power spike-sorting processors. Spike sorting consists of detecting neural spikes in an acquired signal and assigning them to putative neurons in order to distinguish the activity of different sources.

In contrast, the system proposed in this work targets a fundamentally different task, namely seizure detection. While the compared works focus on spike sorting, which involves detection, clustering, and classification, the proposed design emphasizes NCC-based template matching followed by software-side feature extraction and seizure metric computation. As such, the underlying computational models differ significantly. From an architectural perspective, the proposed solution relies on a HW/SW co-design approach, in which the computationally intensive NCC operations are offloaded to dedicated hardware while control and higher-level processing remain on the MCU. In contrast, the referenced works integrate a larger fraction of the signal-processing pipeline directly into dedicated hardware structures.

Despite these differences, the comparison reported in Table 4.4 remains informative at the system level, as it provides a perspective on power consumption across neural signal processing systems. When considering the accelerator alone, the proposed design exhibits a power consumption within the same order of magnitude as the two reference systems, lying between the two comparison points.

When considering the full projected system, the total power consumption reported in this work may appear comparable to that of the compared spike-sorting processors. However, this observation must be interpreted with caution, as it reflects differences in both system architecture and technology nodes. In particular, the proposed design targets a 65 nm process, whereas the OJCAS’2025 system is implemented in a 180 nm technology, making direct power comparisons non-trivial.

Furthermore, this result is consistent with the design choices made in this work, as a significant portion of the processing pipeline remains implemented in software on a general-purpose MCU. In contrast, the referenced works rely on more tightly integrated hardware implementations, resulting in lower overall system power.

	ISCAS'2023 [20]	OJCAS'2025 [21]	This work
Technology (nm)	28	180	65
Clock frequency (MHz)	1.5	150 ^a	1
Core voltage (V)	0.72	1.8	1.8
Input sampling rate (kHz)	40 ^b	30	20
Input word length (bits)	10 ^b	12	12 ^c
Total power (μ W)	20.24 ^d	568.24 ^d	502.73
Accel. power (μ W)	–	–	53.48

^a Derived from the reported cycle count and sampling rate, ^b using the AFE reported in [22], ^c using the AFE reported in [8], ^d reported for 8 channels.

Table 4.4: Comparison with two recent low-power neural spike processing systems. Although the proposed design targets seizure detection rather than spike sorting, the comparison provides a useful system-level perspective on power consumption across embedded neural signal processing architectures. For this work, both the accelerator-only power and the projected total system power are reported.

Conclusion

At the core of this work is the design of a specialized hardware accelerator, together with significant software and algorithmic optimizations, with the objective of building an ultra-low-power vagus nerve-based seizure detection system. In particular, substantial modifications to the ULP VENG-based seizure detection system proposed in [6] are introduced. From an algorithmic standpoint, adaptations are made to better support the deployment of a hardware accelerator for NCC-based template matching. On the software side, the implementation is optimized to reduce power consumption through a reduction of the word length from 32-bit to 16-bit. Most importantly, multiple hardware architectures for NCC-based template matching are explored and compared, ultimately leading to the implementation of a parallel-template NCC hardware accelerator.

Compared to [6], the proposed software-only optimizations enable a **reduction in total execution time by a factor of 4** and a reduction in power consumption by a factor of 3.77 when using a single template. The execution time is reduced to 87.57 ms for a 500 ms window of pre-filtered VENG signal. The measured power consumption is 2.45 mW on the NUCLEO-L4A6ZG development board from STMicroelectronics. As a result, the algorithm can now process **up to 7 templates under real-time constraints**, compared to a single template in [6]. Furthermore, the hardware accelerator-compatible software implementation achieves a reduced power consumption of 1.65 mW at a clock frequency of 24 MHz.

The implementation and logic synthesis of the proposed parallel-template NCC hardware accelerator yield an **estimated total power consumption of 53.48 μ W** at a clock frequency of 1 MHz, while maintaining high computational accuracy, with a median absolute error of 0.748 % for correlation and 0.024 % for NCC. A projection onto the BREWER chip introduced in [8] predicts a system-wide power consumption of 502.37 μ W with a significant portion still attributed to processing operations.

Taken together, these results represent a meaningful step toward the realization of implantable ultra-low-power VENG-based seizure detection systems for drug-resistant epilepsy. In particular, this work demonstrates that NCC-based template matching can be effectively adapted to stringent power constraints through a co-design approach combining algorithmic simplifications, software optimizations, and dedicated hardware acceleration, without compromising detection performance.

Despite these promising results, several limitations must be acknowledged. First, the evaluation of the proposed system relies on a limited set of VENG recordings, which may restrict the generalizability of the results across a broader range of patients and seizure patterns. Second, the evaluation of seizure detection performance through the chosen metric does not provide a complete assessment, as it does not capture aspects such as the spurious detection of epileptic events, detection latency, or other clinically relevant factors. Third, the results obtained for the hardware accelerator are based on post-logic synthesis reports rather than post-physical synthesis or full silicon validation. Consequently, non-negligible deviations in power and area estimates can be expected. Finally, the integration of the hardware accelerator within the complete BREWER system, including the embedded Cortex-M4 processor and the AFE+DBE chain, is not addressed in this work. As a result, additional power and area overheads are anticipated due to the required interfacing with the DBE output, as well as with the system bus and DMA module.

Future work could extend this study in several directions. At the algorithmic level, a first step would involve introducing a periodic template selection mechanism and exploring alternative approaches for peak detection in the aggregated NCC signal. In addition, the seizure detection algorithm should be validated on larger and more diverse datasets, using more comprehensive and clinically relevant evaluation metrics to better assess robustness and generalization capabilities.

From a hardware perspective, accelerating the trace aggregation step could simplify real-time scheduling at the software level while reducing output buffering requirements and improving communication efficiency with the processor. Furthermore, the integration of the hardware accelerator into a complete implantable system-on-chip, including the analog front-end and bus infrastructure, together with full physical synthesis, sign-off, and tape-out, would enable a more comprehensive assessment of system-level performance.

All told, this work demonstrates that a careful co-design of algorithms, software, and hardware architectures enables efficient seizure detection within tight ultra-low-power constraints, thereby bringing implantable VENG-based monitoring systems significantly closer to practical clinical deployment.

Bibliography

- [1] Santiago Martínez et al. “Trends in Volumetric-Energy Efficiency of Implantable Neurostimulators: A Review From a Circuits and Systems Perspective”. In: *IEEE Transactions on Biomedical Circuits and Systems* 17.1 (Feb. 2023), pp. 2–20. DOI: 10.1109/TBCAS.2022.3228895.
- [2] World Health Organization. *Epilepsy: a public health imperative*. Tech. rep. WHO/MSD/MER/19.2. Geneva: World Health Organization, June 2019. URL: <https://www.who.int/publications/i/item/epilepsy-a-public-health-imperative>.
- [3] Patrick Kwan et al. “Definition of drug resistant epilepsy: Consensus proposal by the ad hoc Task Force of the ILAE Commission on Therapeutic Strategies”. In: *Epilepsia* 51.6 (June 2010), pp. 1069–1077. DOI: 10.1111/j.1528-1167.2009.02397.x.
- [4] Myriam Abdennadher, Pratik Rohatgi, and Aneeta Saxena. “Vagus Nerve Stimulation Therapy in Epilepsy: An Overview of Technical and Surgical Method, Patient Selection, and Treatment Outcomes”. In: *Brain Sciences* 14.7 (July 2024), p. 675. DOI: 10.3390/brainsci14070675.
- [5] Lars Stumpp et al. “Vagus Nerve Electroneurogram-Based Detection of Acute Pentylentetrazol Induced Seizures in Rats”. In: *International Journal of Neural Systems* 31.7 (July 2021), p. 2150024. DOI: 10.1142/S0129065721500246.
- [6] Roman Hardy. “Vagus nerve-based seizure detection algorithm analysis and ultra-low-power implementation”. MA thesis. Louvain-la-Neuve, Belgium: UCLouvain, 2024. URL: <https://hdl.handle.net/2078.2/38981>.
- [7] Romain Raffoul et al. “Action Potential Detection Algorithm Adaptable to Individual Nerve and Recording Setup”. In: *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2022, pp. 655–659. DOI: 10.1109/BioCAS54905.2022.9948681.

- [8] Sylvain Favresse et al. “An Ultra-Low-Power Dual-Channel Analog Front End for Vagus Nerve Sensing with Chopping-Minimized Gain Mismatch”. In: *2025 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2025, pp. 121–125. DOI: 10.1109/BioCAS67066.2025.00036.
- [9] Hugo Smets et al. “Analysing vagus nerve spontaneous activity using finite element modelling”. In: *Journal of Neural Engineering* 18.5 (Apr. 2021), p. 056008. DOI: 10.1088/1741-2552/abe68f.
- [10] ARM. *CMSIS-DSP: Overview*. 2026. URL: <https://arm-software.github.io/CMSIS-DSP/latest/index.html> (visited on 04/21/2026).
- [11] Rémi Dekimpe and David Bol. “ECG Arrhythmia Classification on an Ultra-Low-Power Microcontroller”. In: *IEEE Transactions on Biomedical Circuits and Systems* 16.3 (2022), pp. 456–466. DOI: 10.1109/TBCAS.2022.3182159.
- [12] Lars Stumpp. “Vagus nerve electroneurogram for seizure detection in the acute Pentylentetrazol model in rats”. PhD thesis. Louvain-la-Neuve, Belgium: UCLouvain, 2022. URL: <http://hdl.handle.net/2078.1/264708>.
- [13] Michel-Antony Ngan Yamb et al. “Seizure Detection Based on Vagus Nerve Activity: Translation to an Implantable Setting”. In: *2024 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. 2024, pp. 1–6. DOI: 10.1109/MeMeA60663.2024.10596761.
- [14] International Medical Device Regulators Forum. *Good machine learning practice for medical device development: Guiding principles*. Tech. rep. International Medical Device Regulators Forum, Jan. 2025. URL: <https://www.imdrf.org/documents/good-machine-learning-practice-medical-device-development-guiding-principles>.
- [15] U.S. Food and Drug Administration and Health Canada and the United Kingdom’s Medicines and Healthcare products Regulatory Agency. *Transparency for Machine Learning-Enabled Medical Devices: Guiding Principles*. Tech. rep. U.S. Food and Drug Administration, June 2024. URL: <https://www.fda.gov/medical-devices/software-medical-device-samd/transparency-machine-learning-enabled-medical-devices-guiding-principles>.
- [16] U.S. Food and Drug Administration and Health Canada and the United Kingdom’s Medicines and Healthcare products Regulatory Agency. *Predetermined Change Control Plans for Machine Learning-Enabled Medical Devices: Guiding Principles*. Tech. rep. U.S. Food and Drug Administration, Oct. 2023. URL: <https://www.fda.gov/medical-devices/software-medical-device-samd/predetermined-change-control-plans-machine-learning-enabled-medical-devices-guiding-principles>.

- [17] Nonclercq Antoine. *BEAMS-Biomechatronics/AP-Detection-Algorithm*. Jan. 2024. URL: <https://github.com/BEAMS-Biomechatronics/AP-Detection-Algorithm> (visited on 04/21/2026).
- [18] Pol Maistriaux, Jérôme Louveaux, and David Bol. “Optimization and Integration of Low-Area Low-Energy and High-SQNR FFT Accelerators in ULP MCUs”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2026), pp. 1–13. DOI: 10.1109/TCSI.2026.3664506.
- [19] STMicroelectronics. *Ultra-low-power Arm[®] Cortex[®]-M4 32-bit MCU+FPU, 100DMIPS, 1MB Flash, 320KB SRAM, USB OTG FS, audio, AES+HASH, ext. SMPS*. STM32L4A6xG datasheet. July 2022.
- [20] Hao Jiang et al. “A 2.53 μ W/channel Event-Driven Neural Spike Sorting Processor with Sparsity-Aware Computing-In-Memory Macros”. In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2023, pp. 1–5. DOI: 10.1109/ISCAS46773.2023.10181615.
- [21] Yuntao Han et al. “L-Sort: On-Chip Spike Sorting With Efficient Median-of-Median Detection and Localization-Based Clustering”. In: *IEEE Open Journal of Circuits and Systems* 6 (2025), pp. 205–216. DOI: 10.1109/OJCAS.2025.3584317.
- [22] Liangjian Lyu et al. “A Fully-Integrated 64-Channel Wireless Neural Interfacing SoC Achieving 110 dB AFE PSRR and Supporting 54 Mb/s Symbol Rate, Meter-Range Wireless Data Transmission”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.5 (2020), pp. 831–835. DOI: 10.1109/TCSII.2020.2982208.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl