HD Recall of Reactive Behavior through Multi-Modal Sensor Fusion

Alisha Menon

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-13
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-13.html

May 1, 2022

HD Recall of Reactive Behavior through Multi-Modal Sensor Fusion

By Alisha Menon

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Jan Rabaey
Research Advisor

May 29, 209

(Date)

* * * * * * *

[illegible]

Professor Rikky Muller
Second Reader

05/29/20

(Date)

Abstract

HDC Recall of Reactive Behavior through Multi-Modal Sensor Fusion

by

Alisha Menon
Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley
Professor Jan Rabaey, Chair

To address prosthetic applications and the issues that cause users to abandon them, particularly to lessen the burden of prosthetic control on users, this work aims to implement the concept of “shared control” for prosthetic reflex task response. The proposed algorithm, Hyperdimensional Computing (HDC), has previously been demonstrated to recall reactive behavior [1] and also improves sensor fusion performance by inherently binding features extracted from arbitrary data streams. Towards exploring HDC sensor fusion, both early and late sensor fusion inference ASICs are synthesized in the TSMC 28HPM process and various optimizations to the algorithm are proposed and evaluated that have significant implications on multi-modal fusion power and area requirements. Additionally, vectorization of the many-channel HDC architecture is explored through implementation on the Rocketchip RISC-V processor [2] by optimizing an existing C implementation of HDC inference [3]. A strategic combination of algorithm tweaks, code optimizations, and vector acceleration using the Hwacha vector processor [4] yielded multiple orders of magnitude (~200×) speedups for HDC classification compared to [3]. Towards the goal of developing a platform to explore HDC recall of reactive behavior for prosthetic applications, force sensors have been selected and characterized, and a PCB has been fabricated to integrate the force sensors into the existing EMG classification adapter board [5]. The proposed algorithm was demonstrated to be well-suited for sensor fusion tasks leaving exciting avenues for future work in integrating the sensor fusion properties with the recall of reactive behavior platform.

Acknowledgements

I would like to thank the following people who contributed to this work: Harrison Liew with whom the HDC sensor fusion ASIC syntheses and CPU optimization/vectorization was jointly done, Professor Krste Asanovic who provided invaluable feedback on the bit-serial word-parallel technique used in the CPU optimized spatial encoder and Professor Sophia Shao who helped shape the direction of the sensor fusion point exploration work.

Professor Jan Rabaey has my deepest appreciation for providing me with guidance for the past 4 years. My research career in engineering began thanks to him giving an opportunity to an undergraduate student who had only interest and little experience. I would not be where I am today without his kindness and support. Thank you to Professor Rikky Muller who has continuously been beyond generous with her time and mentorship. Thank you to Ali Moin and Andy Zhou who have been a constant resource and provided advice throughout my time at UC Berkeley.

I want to express my sincere gratitude towards Rozhan Rabbani and Rebekah Zhao; I am privileged to have been able to walk on this journey and make precious memories over the past few years with two incredibly strong friends by my side.

Thank you to KNJ, KSJ, MYG, JHS, PJM, KTH and JJK for inspiring me everyday to reach higher, to see the ups and downs in a new light, and to become a person who stands confidently on her own.

Lastly my family, to whom I am forever indebted. Words can't describe how much I love and owe you for always supporting me, encouraging me and believing in me more than anyone else, including myself. Since leaving home two years ago I have learned so much from so many people, but even with the distance I always learn the most from you three who give me the tools and love to live my life with pride and without fear.

Chapter 1

Introduction

Existing prosthetic systems face difficulty in accurately and quickly accomplishing reflex tasks which would require little to no effort for a human arm. To address prosthetic applications and the issues that cause users to abandon them, this work contributes towards the goal of lessening the burden of prosthetic control on users by exploring the concept of "shared control." This involves combining user input and a trained controller to achieve an objective. Through the integration of information from force or motion sensors in addition to the physiological signals (e.g. EMG) used to decode intention, low-level reflex controls are transferred from the user to the prosthesis.

Previous work in this area [1] has attempted to implement shared control through the use of an multilayer perceptron (MLP) for myoelectric decoding and a previously developed robotic controller to maximize area of contact between the prosthetic and an object to maintain grip. In [1] however, the controller assumes full control once contact with an object is made until all contact points are met, removing the user's ability to manipulate the prosthetic during that time. Additionally, this approach is applicable only to paradigms in which contact should be maximized such as grasping and can't be expanded to other reflex actions. [2] developed a method for context-based prosthetic grasping where a variety of sensors are used to to determine the state of the grasping action (free, preshaped, grasping, holding, and moving). Different rules are applied to the myoelectric controller output depending on the current state such as modulating wrist rotation and hand opening thresholds to reduce the impact of accidental rotations or openings during grasping. This system uses additional sensors to decrease the effect of potential errors in intent decoding, but does not actually assist in task performance; the burden of completing the action through prosthetic control remains completely on the user. However, it does demonstrate an interesting concept to build into a shared control scheme.

A unique reflex control is seen in [3] which primarily focused on texture classification and implements a control loop to detect "pain". At recognition of a sharp object, the robotic hand automatically releases grip on an object. This is an example of another reflex reaction that benefits from shared control. Additional work [4] uses a neuromimetic tactile sensing system to detect the onset and offset of object contact. Using that information, the level of user control through EMG is modulated to reduce the effect of misclassifications on object grasping. Changes in sensor information trigger object slip control implemented through directly increasing prosthetic grip in a quantized manner. Though [4] does technically share control between the prosthetic and the user, the technique is still only applicable to the problem of object grasping.

To achieve a shared control system that equally integrates user intent with a controller to truly share the performance of a variety of reflex tasks, not just limited to object grip or release, this work proposes to use Hyperdimensional computing (HDC), an area of active research that is used very effectively for classifying physiological signals. It is based on the idea that human brains do not perform inference tasks using scalar arithmetic, but rather manipulate large patterns of neural activity. These patterns of information are encoded in binary HVs, with dimensions ranging in the 1,000's to ensure that any two random HVs are completely orthogonal to each other. There are three operations that are performed on these HVs: bundling, binding, and permutation. Bundling is a component-wise add operation across input vectors, binding is a component-wise XOR operation, and permutation is a 1-bit cyclical shift. The simplicity of these operations suggests and previous work [5] [6] has shown that HDC is very computationally efficient.

HDC has been preliminarily demonstrated in previous work [7] to be able to encode reactive behavior for later recall, specifically for a navigation task. Through integration of force sensors into a previously developed HDC EMG classification system [8], this work proposes to use this technique to accomplish prosthetic shared control. HDC encodes information in the same form no matter the complexity of the sensor inputs; this lends itself well to the sensor fusion aspect. Using this property, the bundling operation to create a vector similar to the inputs and the binding operation to create a vector different from the inputs, but from which each input can be recoverable, the force sensors, EMG signals and actuator (prosthetic) signals are fused to create a set of vectors over the course of the training period that are bundled together and stored in the "program memory" as a single

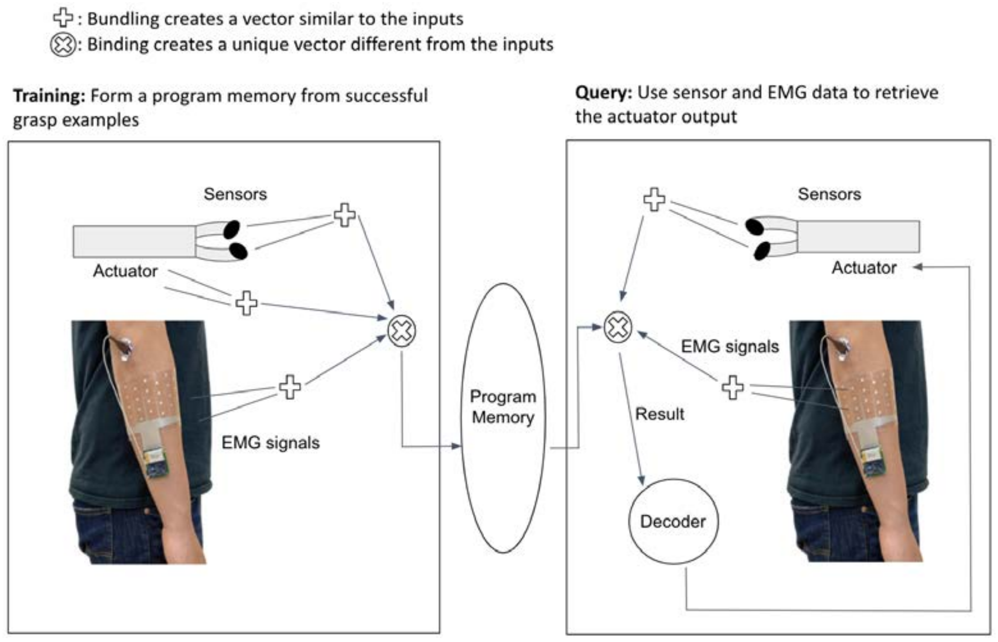

: Bundling creates a vector similar to the inputs
: Binding creates a unique vector different from the inputs
Training: Form a program memory from successful grasp examples
Query: Use sensor and EMG data to retrieve the actuator output
Sensors
Actuator
EMG signals
Program Memory
Result
Decoder
Sensors
Actuator
EMG signals

Figure 1.1. HDC recall of reactive behavior technique applied to prosthetic reflex using EMG and force sensors

HV. Then, using EMG signals and force sensors, the resulting actuator vectors can be later decoded/recalled from the program memory by unbinding them from the sensor-actuator pairs. An associative memory is used to decode and return a cleaned up actuator signal to the prosthetic to react to sensor events. This technique is shown in Figure 1.1; the method has the potential to become a very powerful tool for recalling various reflex tasks, while still maintaining the user's control over the system through integration of EMG sensors and force sensors.

Towards the sensor fusion aspect of shared control, neural networks (NNs) have been the subject of intense research, but there has not been significant results in optimizing training/inference with multi-sensor inputs. NNs are often comprised of many layers (convolutional, fully-connected, etc.), so the natural question arises: at which point should sensor data be fused? Prior work [9] evaluated the performance of a object detector using a camera + light detection and ranging (Lidar) as input. The underlying feature encoder is based on VGG16 (5-layer convolutional NN) with classification generated using an MLP. Variations of this network performing the camera/Lidar sensor fusion early as in Figure 1.2, late as in Figure 1.3, or in the middle are evaluated, showing that the later the sensor data is fused, the better the object detection rate, but at the expense of multiple parallel convolutional layers. Essentially, this implies that it is difficult to parameterize a NN sufficiently to extract all the features from early-fused sensor inputs to achieve equal accuracy

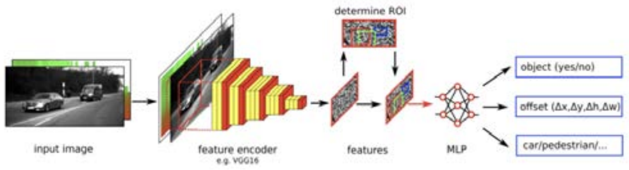


Figure 1.2. Early fusion architecture implemented in [9]

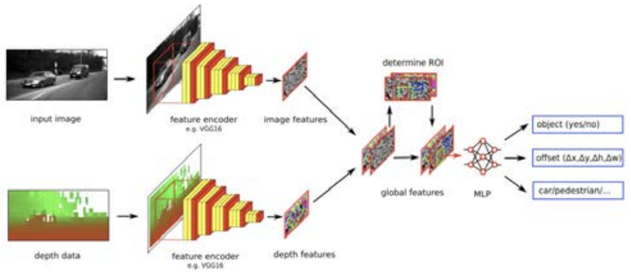


Figure 1.3. Late fusion architecture implemented in [9]

as compared to a NN where the already extracted features are being fused. This represents a trade-off between complexity & accuracy - early fusion degrades accuracy because of unintelligent feature extraction, unless compensated by larger nets while late fusion requires parallel networks up to the fusion point.

In contrast, HDC inherently binds features extracted from arbitrary data streams without exceeding the capacity of HVs, suggesting early fusion will have less effect on accuracy using this algorithm, breaking the complexity-accuracy tradeoff. In addition, using a spatial encoder to map channels and feature values into HVs, a temporal encoder to map changes over N samples, and an associative memory (AM) to store classes and conduct inference, HDC offers a reduction of computation and memory requirements, ability to use the same hardware for training & inference, rapid training time, and robustness to noise/variations in input data making it a viable choice for embedded sensor fusion applications. In this work, the accuracy and hardware implications of the sensor fusion point for HDC are explored through synthesizing an ASIC for both early and late fusion architectures (details in Chapter 3.2 & Chapter 4.3), with additional work on optimizations specific to multi-modal applications.

In terms of hardware acceleration, the fundamental operations underneath the many layers in a NN (convolutional, fully-connected, etc.) are multiply-and-accumulates on tensors in floating-point for training, potentially low-precision integer for inference. Accelerating NN training has benefited greatly from existing SIMD architectures such as GPUs, while inference has been accelerated further using specialized hardware implementing the gen-

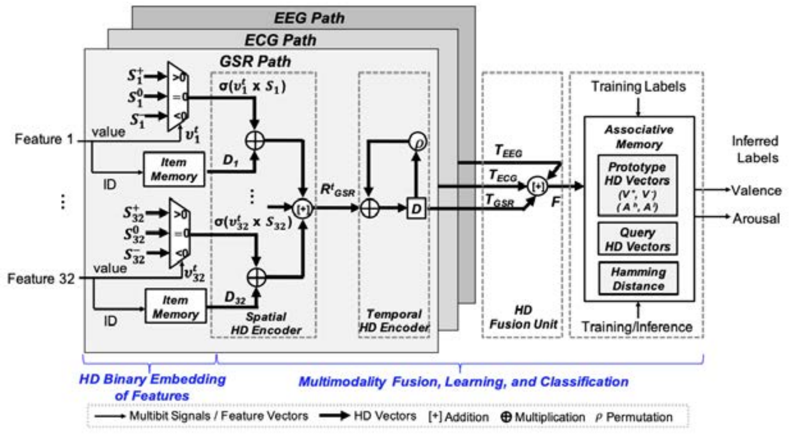


Figure 1.4. HDC sensor late fusion architecture datapath [10]

eral matrix multiplies (GEMM) in systolic arrays. In comparison, HDC for both training and inference can be accelerated on almost any existing computing architecture because of the simplicity of its binary bit-wise operations which are present in one form or another on all hardware platforms, and is further resilient to slight changes to the algorithm for platform-specific optimizations as long as the orthogonality of HVs is maintained. Because of the basic binary operations conducted on the very-large hypervectors, it was hypothesized that a vector accelerator would be able to hugely impact the performance in terms of cycle count of the HDC algorithm, specifically in cases with very large numbers of channels as in sensor fusion applications. Therefore, in addition to the early/late fusion hardware trade-offs demonstrated through ASIC synthesis, the vectorizability of HDC for sensor fusion was explored deeply in this work on the Rocketchip RISC-V CPU with the Hwacha vector accelerator.

The algorithm in [10] is used as the basis for optimizations and evaluation of HDC sensor fusion. In this algorithm, features from three different modalities are fused together into one HV that can be used for learning and inference of a classification outcome. The method of fusion occurs only before the associative memory which results in multiple parallel spatial/temporal encoders for each sensor modality. Using this algorithm, the paper demonstrated better accuracy than SVM, XGB, and Gaussian NB for all amounts of training data. The HDC algorithm does not show significant degradation in accuracy as the amount of training data decreases even down to 20% while other algorithms' performances slowly degrade. Figure 1.4 shows the late fusion architecture used in [10] for multi-modal sensor fusion which contains a spatial encoder and temporal encoder for each modality. These are bundled together to form a query HV used for inference in the AM.

To evaluate the impact of the sensor fusion point on the accuracy of HDC classification, Chapter 2 details the original algorithm used in [10] and proposes a modification to bundle HVs across the various modalities before the temporal encoder instead of after it. Chapter 3 describes prior work in mapping other HDC algorithms to various hardware platforms providing evidence of HDC computational efficiency and a baseline for comparison for the two target platforms in this work: ASIC and Rocketchip RISC-V CPU with the Hwacha vector accelerator.

Chapter 4.1 describes the modifications to [10] for both early and late fusion for the multi-modality dataset, and the algorithm design and parameter selection process for optimal accuracy. Targeting an ASIC, Chapter 4.2 elaborates on the subsequent translation to RTL. Chapter 4.3 details the synthesis efforts in the TSMC 28HPM process. Chapter 4.4 analyzes the accuracy of the algorithm and its area and power requirements, and Chapter 4.5 identifies and evaluates specific areas for further optimizations for multi-modal sensor fusion applications.

Targeting a vector accelerated Rocketchip RISC-V CPU, Chapter 5.1 describes the translation of the designed early fusion HDC architecture to C code, optimizations to target the open-source Rocketchip RISC-V CPU, and translation to vector assembly to utilize the Hwacha vector accelerator. Chapter 5.2 details the simulations that were run on both the Rocket-only and the combined Rocket + Hwacha systems and profiling methods. Chapter 5.3 analyzes the speedups obtained with each optimization implemented, and Chapter 5.4 identifies areas for further optimizations.

The last part of this work returns to the HDC recall of reactive behavior technique. Chapter 6 elaborates on the steps taken towards demonstrating proof-of-concept for this technique. Finally, it is concluded in Chapter 7 that HDC algorithms are well-suited for early sensor fusion as well as acceleration on vector machines with exciting avenues for further research efforts in exploring optimizations specifically targeted towards multi-modal classification tasks based on the results found in this work and in integrating the demonstrated HDC sensor fusion techniques with the developed HDC recall of reactive behavior platform.

Chapter 2

Multi-Modal Dataset & Algorithm

2.1 AMIGOS Dataset

Previous work [10] on HDC for multi-modal physiological signals used the AMIGOS dataset developed by [11]. In this study, GSR, ECG and EEG signals were measured for 33 subjects as they watched 16 videos. Each video for each subject was classified to have either led to a positive or negative emotion (valence), and the strength of the emotion was classified as either strong or weak (arousal). From the 3 modalities, [12] carefully selected 214 features relevant to an accurate classification. GSR has 32 features, ECG has 77 features, and EEG has 105 features. The 1,467 seconds of data for all 33 subjects were appended and a moving average of 15 seconds over 30 seconds was applied resulting in a total of 3,036 samples. This was further downsampled by a factor of 8 for more rapid processing and usage of the HDC classification algorithm.

2.2 HDC-MER Architecture

The original HDC multi-modal sensor fusion architecture in [10] includes three main blocks. The spatial encoder projects feature channel information into the hyperdimensional space through the use of a randomly generated binary channel HV for each of the 214 features. For this implementation, a HV dimension of 10,000 is used for the full datapath. The channel HVs are stored in an item memory (iM). In order to encode feature values, an additional feature projection vector {-1,0,1} is randomly generated for each channel and

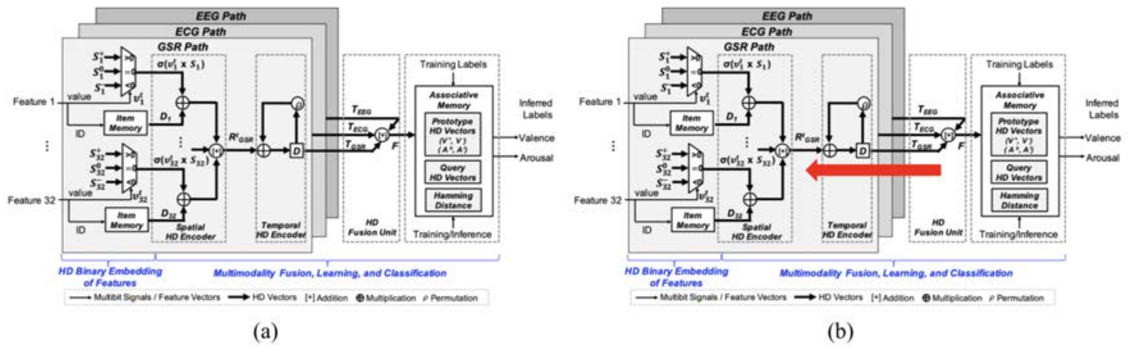


Figure 2.1. HDC sensor fusion architecture [10] (a) previous work's implementation (b) proposed modification for early fusion

stored in the feature projection memory. This vector is then multiplied by the feature value and binarized.

The binding operation is utilized to combine the feature projection HV with the channel HV, generating a spatially encoded HV for that channel. To develop a complete HV for each modality, the bundling operation combines the many spatially encoded HVs. To complete the encoding process and include information about temporal changes, the permutation operation is used to keep track of previous samples. HVs coming from the spatial encoder are permuted and then bound with the next HV N times in the temporal encoder. This results in an output that observes changes over time.

In order to combine the temporally encoded vectors from different modalities, a final bundling operation is used. During the training process, many such encoded HV are generated, bundled to represent a class and then stored into the final block, the associative memory. During inference, the encoded HV is compared against each trained HV using Hamming distance. For binary vectors, this involves an XOR and then popcount. The comparison with least distance is the inferred label. For this dataset, there are two classifications to be made, one for strength of emotion and one for polarity of emotion, resulting in two separate AMs.

2.3 Early/Late Fusion

The HDC-MER architecture implements "late fusion" in that the fusing of the three physiological modalities occurs only after the temporal encoder. This results in three parallel encoding datapaths. Because HDC binds information streams through encoding in large, orthogonal vectors with a very large capacity, it was hypothesized that fusing at an earlier point would not result in any significant classification accuracy degradation. Therefore, in this work, an “early fusion” scheme is proposed in which the fusion point occurs after the spatial encoder as opposed to after the temporal encoder as shown in Figure 2.1. This eliminates the need for two temporal encoders, thereby reducing the necessary blocks down to three spatial encoders, a temporal encoder and an AM.

Chapter 3

State of the Art Hardware Implementations

Much of the algorithm development in HDC research has been implemented in MATLAB, which is great for prototyping but inefficiently encodes every ”bit” in the HVs as distinct floating-point numbers. To date, there have been some efforts to map HDC to CPUs, GPUs, FPGAs, and custom hardware. These previous hardware implementations and comparisons have demonstrated the benefits of HDC for other applications, providing a baseline for comparison of this work.

3.1 Voice-HD

In [13], HDC was implemented and evaluated for efficient speech recognition. The HDC algorithm was benchmarked and compared against a DNN with 3 hidden layers on CPU cores using Intel Core i7 processor with 16 GB memory (4-core, 2.8GHz) for training and testing. HDC was shown to recognize voices with similar accuracy as DNNs but with 11.9$\times$ higher energy efficiency and 5.3$\times$ faster testing time. This provides evidence of the hardware efficiency of HDC as compared to other ML algorithms and also introduces motivation for exploration of HDC hardware accelerators to test the performance limits and bottlenecks.

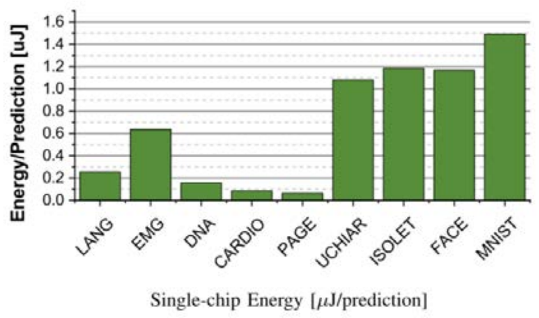
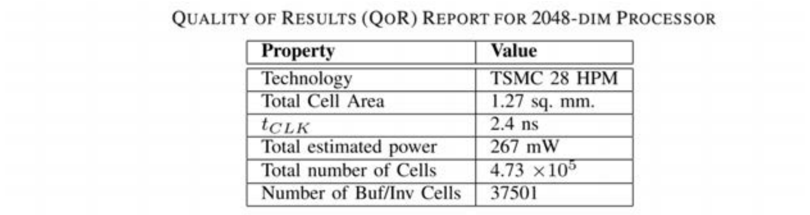


Figure 3.1. ASIC Energy per prediction for various classification tasks from [6]



Quality of Results (QoR) Report for 2048-dim Processor

Property	Value
Technology	TSMC 28 HPM
Total Cell Area	1.27 sq. mm.
t_{CLK}	2.4 ns
Total estimated power	267 mW
Total number of Cells	4.73×10^5
Number of Buf/Inv Cells	37501

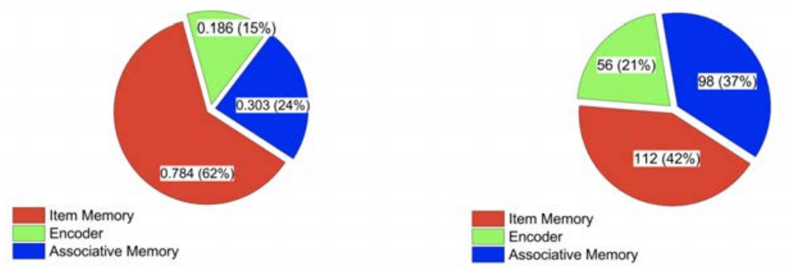


Figure 3.2. ASIC QOR results and block breakdown (right: area, left: power) from [6]

3.2 Generic, programmable and scalable HD architecture

A generic, programmable, and scalable HD architecture on CPU, eGPU, and a synthesized ASIC design in a 28nm HK/MG process are presented in [6]. The results are shown in Figure 3.1 & Figure 3.2 for a 2048-dimension HV HDC ASIC processor. The processor was scalable up to 256 channels, had an AM with 8-cycle latency to reduce the timing critical path, and was synthesized in TSMC 28HPM with the iM compiled into ROM. The results from [6] show that the ASIC was dominated by the item memory for area and dominated by both the item memory and the associative memory for power. This accelerator doesn't provide the architecture for multiple modalities, thus leaving room for exploration and optimization of HDC ASICs for sensor fusion, specifically regarding early and late fusion, but it does provides a baseline for the work done in this project.

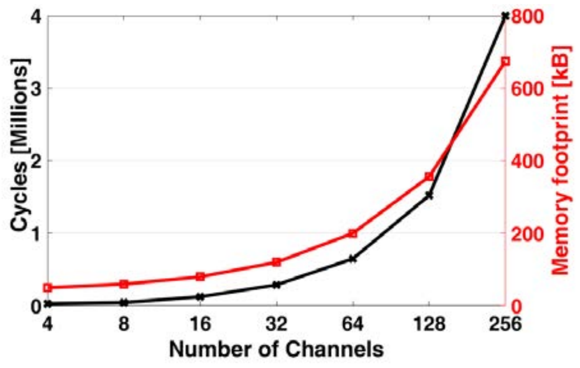


Figure 3.3. PULP-HD performance and memory footprint vs. input channels [5]

3.3 PULP-HD

Efforts to accelerate a generic HDC algorithm using two generations of the parallel ultra-low power (PULP) platform [14] (an embedded SoC that exploits near-threshold operation and multi-core processing to accelerate many tasks including signal processing applications) are presented in [5]. Compared to a single ARM Cortex M4, the algorithm on the PULPv3 platform achieved 3.7× speedup using 4 cores and on the Wolf platform which employs 8 cores and special bit manipulation instructions [15]) achieved a total 18.4× speedup.

The major optimizations the authors applied in the process of mapping from MATLAB were the packing of HVs across many 32-bit unsigned integers, multi-core processing, and data partitioning between L1$ and L2$. The work also analyzes the impact on the memory footprint and load cycles for various different HDC parameters, such as the HV dimension D, number of channels, and temporal encoder size N. An interesting analysis is shown in Fig. 3.3, where the algorithm's footprint scales super-linearly with the number of input channels. Since the algorithm chosen for the multi-modal sensor fusion application in this work has a 214 input channels, this is a prime area for optimization.

3.4 Optimization Techniques

Additional relevant work [16] proposes hardware techniques to optimize HDC in a open-source VHDL library. The goal was to co-locate learning and classification tasks on a small portion of Xilinx UltraScale FPGAs via tricks such as rematerializing HVs on the fly to save storage, counter-less binarized back-to-back bundling, and single-cycle associative memories. The optimizations save 2.39× in area and 2,337× in throughput. The Pareto optimal HD architecture to learn and classify five hand gestures using four electromyography sensors is mapped on only 18,340 configurable logic blocks (CLBs).

Chapter 4

HDC Sensor Fusion ASICs

4.1 Algorithm

The HDC algorithm was first designed in MATLAB, building upon the original HDC-MER architecture. The "early fusion" scheme was implemented to determine accuracy performance as compared to the original late fusion scheme. The resulting overall mean classification accuracy is shown in Figure 4.1. These results are very encouraging as they demonstrate that, not only does early fusion perform as well as late fusion, it actually performs better than late fusion for larger values of N in the temporal encoder (observing changes over a larger number of samples). This is hypothesized to be a result of the temporal encoder's effect on each modality. Fusing after the temporal encoder results in selection of N that may not be optimal for each modality, just best overall performance. Fusing before the temporal encoder allows each modality to be encoded temporally together which could

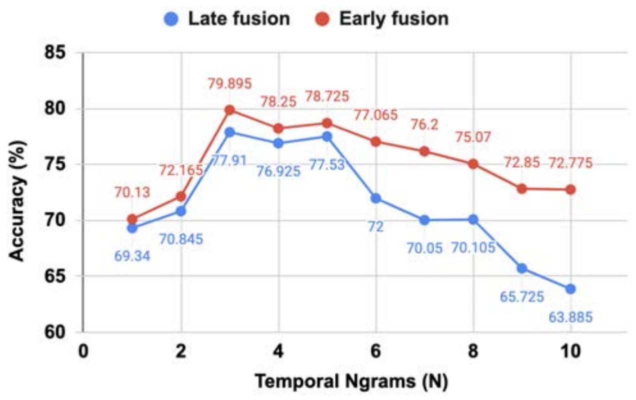


Figure 4.1. Late and early fusion mean accuracy vs. temporal Ngrams

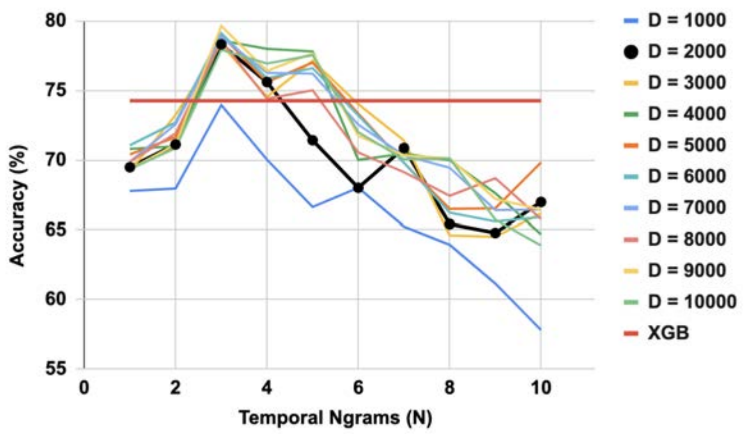


Figure 4.2. Change in late fusion mean accuracy as hypervector dimension decreases compared to the known best alternative (XGB) vs. temporal Ngrams

lessen the impact of temporal Ngrams on specific modalities, improving the overall accuracy across all N values.

Based on these results, $N = 3$ was selected as the optimal for mean accuracy. For this dataset that includes signals of various relevant frequencies (GSR the slowest at [0 - 2.4] Hz, ECG at [0 - 6] Hz and EEG the fastest at [2 - 47] Hz), $N = 3$ may be the optimal point due to the tradeoff between observing larger periods for the slower signals and smaller periods for the faster signals. Specifically in this case, since EEG has the largest number of features (105 EEG features compared to 77 for ECG and 32 for GSR), perhaps the algorithm performs better when observing smaller time periods to track faster changes from the faster signals like EEG which can help to better represent the classes. Future work could include a ”parameter-less” temporal encoder that does not require this parameter selection beforehand, making it more generalizeable to different datasets.

In conclusion, as suspected, early fusion for HDC does perform just as well as late fusion, breaking the trade-off between early fusion and accuracy that traditional ML methods face.

4.2 ASIC Implementation

4.2.1 Algorithm Optimization

In order to optimize for hardware, a few preliminary changes were made to decrease the number of necessary operations and memory. The first and most significant was the dimension of the HVs. The original work was done using $D = 10,000$ for which random HV

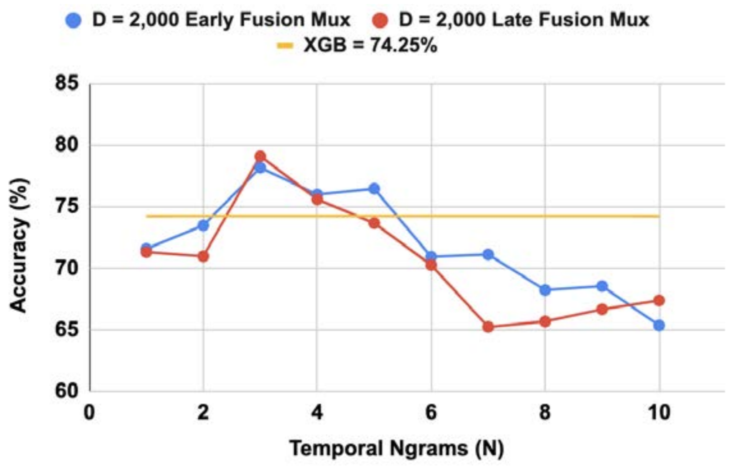


Figure 4.3. Mean accuracy using the same iM per modality, D = 2,000 and multiplexed feature projection vectors for early and late fusion compared to known best alternative (XGB) vs. temporal Ngrams

pseudo-orthogonality is sufficiently high for the purposes of the AMIGOS dataset. As the dimension decreases, the pseudo-orthogonality also decreases leading to greater unintentional clustering hence impacting accuracy. This is demonstrated in Figure 4.2 which shows the accuracy degradation that occurs as D decreases. The baseline in the figure represents the performance of the known best alternative classification algorithm, extreme gradient boosting (XGB). The smallest dimension that still performs better than the baseline was $D = 2,000$ at $N = 3$.

The next optimization that was made was using the same iM for each modality which led to no performance degradation, likely due to the fact that the feature projection vectors for each channel were still different resulting in there still being 214 different encoded vectors for each feature. Another preliminary optimization made was using a multiplexer for the feature projection vectors. In the original algorithm, the feature projection vector {-1, 0, 1} for each of the 214 channels is multiplied by the feature value and then binarized by reducing the positive values in the vector to 1s, and the zeros and negative values to 0s. This process can be simplified to selecting between a pre-generated and pre-multiplied negative or positive feature projection vector depending on the feature value's sign. The optimization resulted in the elimination of multipliers which would otherwise have been very costly. The result of these changes is shown in Figure 4.3 to still perform better than the known best alternative classification method for $N = 3$.

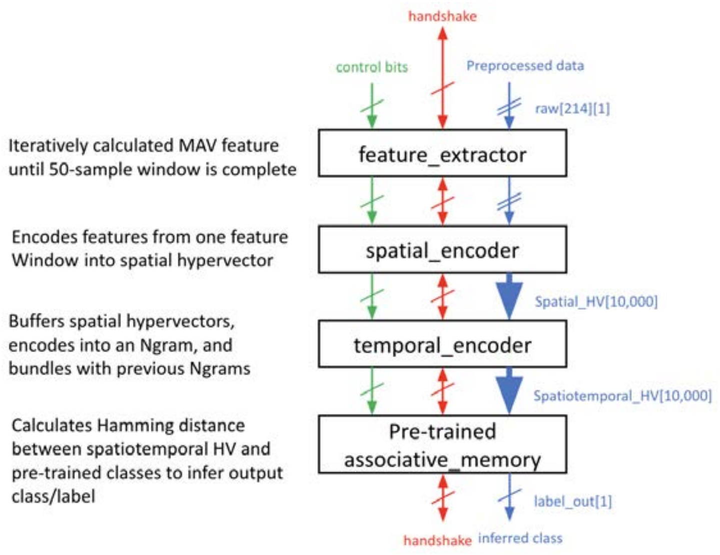

handshake
control bits
Preprocessed data
raw[214][1]
Iteratively calculated MAV feature until 50-sample window is complete
feature_extractor
Encodes features from one feature Window into spatial hypervector
spatial_encoder
Spatial_HV[10,000]
Buffers spatial hypervectors, encodes into an Ngram, and bundles with previous Ngrams
temporal_encoder
Spatiotemporal_HV[10,000]
Calculates Hamming distance between spatiotemporal HV and pre-trained classes to infer output class/label
Pre-trained associative_memory
label_out[1]
handshake
inferred class

Figure 4.4. Verilog HDC for sensor fusion built on previous architecture for EMG

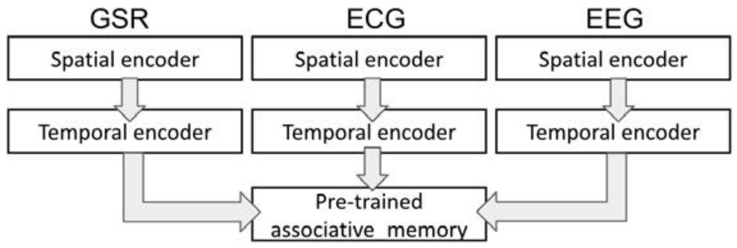


Figure 4.5. Late Fusion Structure

4.2.2 RTL: Verilog

The HDC multi-modal sensor fusion algorithm that was designed was then written in Verilog for the purpose of synthesizing an ASIC. Previous Verilog code has been written for 64-channel HDC EMG gesture classification [8] based on [16] which demonstrates optimization of the basic HDC algorithm blocks for FPGA implementation and includes models for pipelining and parallelism for the spatial encoder, the temporal encoder, and the associative memory that provide a tradeoff between latency and energy/prediction.

The sensor fusion architecture was built on top of a ready/valid handshake protocol

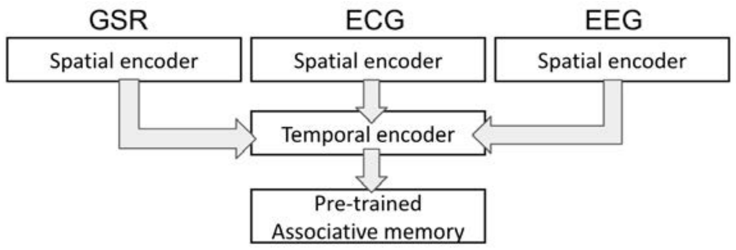


Figure 4.6. Early fusion structure

between the different blocks in the datapath as shown in Figure 4.4. In order to modify the RTL for this application, the feature encoder was removed and, since this project focuses on inference, the AM was pre-trained, feature values were pre-processed, and all training mode logic was removed. In addition, the temporal encoder was modified for the selected $N = 3$. Then, to include the sensor fusion aspect, parallel paths for early and late fusion of multiple data modalities were implemented resulting in the structures shown in Figure 4.5 and Figure 2.1.

4.2.3 RTL: Chisel

The Verilog RTL as written only has the logic to process the HVs, but does not store the HVs needed to implement the item and feature projection HV memories. To obtain accurate power and area numbers, real memories were required, so a wrapper module was written in the Chisel hardware description language with the Verilog RTL blackboxed. In Chisel, abstract sequential memory constructs can be described very easily to be later mapped down to technology SRAMs. This wrapper module is written as a memory-mapped peripheral block to the open-source Rocketchip RISC-V core and can eventually be expanded to generate an SoC with an HDC accelerator attached.

4.3 Synthesis

The Chisel wrapper and Verilog are elaborated in the Chipyard SoC generator [17], and the memory mapping and synthesis flows were rapidly developed using the Hammer framework included within. At first, the Intel 22FFL technology was selected due to a mature Hammer flow through place-and-route; however, it was quickly discovered that the required memory sizes were not present in this technology. Specifically, the item and feature projection HV memories need to be of size $channels$ deep and D wide.

Unfortunately, the minimum depth of Intel 22FFL's SRAMs was 512, meaning many rows would be wasted. Therefore, the TSMC 28HPM was selected for synthesis instead, which has the requisite SRAM ranges for this use case. The metrics of interest for comparison are power and area. Post-synthesis simulations were not yet run on the design, so the power numbers are not workload-accurate, future work could include FPGA synthesis for prior testing. Since the focus is on the difference between early and late fusion, the absolute number is less important for the analysis than the difference and breakdowns by block.

4.4 ASIC Results

4.4.1 Accuracy

Initial partial synthesis results demonstrated a clearly overwhelming contribution to the overall area and power by the SRAMs. To address this before completing a full synthesis, further optimization was explored on the algorithm side as to how best to decrease the necessary memory. Previous optimization involved using the same iMs for each modality, another obvious choice was to use the same iMs and projMs for both classifications being made (strength of emotion - arousal, and polarity of emotion - valence). Since this would still result in random vectors for each channel, in theory this would have no affect on the performance. Pushing further, using even the same feature projection vectors for each modality was explored. This would result in both the positive and negative feature projection vector memory being reduced by 109 HVs out of the original 214 channels with the limiting factor being the 105 channels in EEG, the largest modality.

Mean accuracy	Early Fusion	Late Fusion
(1) Shared modality iM	78.17%	79.10%
(2) A&V shared iM & projM	79.77%	78.70%
(3) Shared modality iM & projM	78.18%	79.24%
(4) Best known alt.	74.25%	74.25%

Table 4.1. Accuracy performance of HDC early and late fusion for: (1) a shared iM across each modality, (2) in addition using the same iM and projMs for both valence and arousal classification, (3) in addition using the same projMs across each modality, (4) the best known alternative classification method (XGB).

Both of these changes were implemented in MATLAB for $D = 2,000$ in addition to all previous optimizations. Since using the same feature projection vectors (projMs) led to inconsequential changes in mean accuracy (early fusion accuracy decreased by 1.6% while late fusion increased by 0.5%), HDC was still consistently around 4% better than the next best alternative as shown in Table 4.1.

In theory, the re-used feature projection vectors might lead to some correlation between specific channels in each modality, but the extent of that for high-dimensional vectors was not enough to result in any significant loss in performance on this specific dataset. This suggests that either the randomness was sufficient enough that resulting accuracy changes are due to chance, or that this dataset is resistant to correlation between modalities. In terms of a tradeoff, the benefits of the 50.93% memory reduction far outweighed the slight early fusion accuracy degradation that still maintains HDC's top performance on AMIGOS.

The final architecture for which synthesis was completed included all previous optimizations: $D = 2,000$, multiplexed feature projection vectors, shared iM and feature projection vectors across each modality, and the same iM and feature projection vectors used for both strength of emotion and polarity of emotion classifications.

4.4.2 Power & Area

The results from synthesis on the RTL implementing both early and late fusion are shown in Table 4.2. At a 3ns clock period, the early fusion version has 6.2% lower power and 4.3% lower area than the early fusion version. Digging deeper into the reports, it was found that despite sharing the memories between the modalities, the memories are still a very sizable portion of the power and area requirements. The breakdown of the area and power utilization for each hierarchical block in both versions are shown in Tables 4.3 and 4.4, respectively. It can easily be calculated that without sharing the memories, the area requirement for memories alone would balloon to 72% for the early fusion version.

Metric	Late Fusion	Early Fusion	Difference
Instances	398253	381294	-4.3%
Total Area (mm^2)	0.932	0.893	-4.3%
Total est. power (mW)	334.5	313.6	-6.2%

Table 4.2. Early vs. Late Fusion Synthesis Results

Block	Early Fusion	Late Fusion
Spatial encoder	48.9%	46.4%
Temporal encoder	2.0%	5.8%
Associative mem	1.2%	1.5%
Item & Proj. mems	47%	45.1%

Table 4.3. Area Breakdown

Block	Early Fusion	Late Fusion
Spatial encoder	42.7%	39.6%
Temporal encoder	3.1%	8.0%
Associative mem	2.3%	3.5%
Item & Proj. mems	49.9%	47.0%

Table 4.4. Power Breakdown

An interesting observation is that for HDC classification tasks, the item and feature projection HV memories need to be identical between training and inference. Furthermore, they are mostly collections of orthogonal HVs, and not dependent on the classification task, as long as there are enough HVs to cover the number of channels for each modality. Therefore, these memories could be implemented using technology non-volatile ROMs. Unfortunately, the Chipyard infrastructure does not allow compiling down to technology ROMs like it can to SRAMs, so Table 4.5 describes the differences between ROMs and SRAMs of equivalent size (the ones this work's algorithm require) for the TSMC 28HPM technology.

It is clear that using ROMs provides significant reductions in area and power at the expense of higher leakage power and longer clock cycles. These are amplified as the size of the memory increases, corresponding to larger numbers of channels per modality. Notably, since HDC classifiers are often used for slow biological signals, the leakage power may be of more concern than timing closure. In another technology, the ROM leakage power and cycle time may perform better; there is room for further exploration on technology selection based on its ROM performance, especially for HDC sensor fusion applications in which the memory dominates the power and area of the ASICs.

Mem. Size	Area	Dyn. Power	Leak. Power	Cycle Time
32x125	-31%	-18%	+131%	+133%
128x125	-39%	-20%	+170%	+133%

Table 4.5. ROM vs. SRAM comparison

Further analysis of the reasons for why the difference between early and late fusion is so small shows that this stems from the fact that performing temporal binding is not an expensive operation: all that is necessary is a couple of registers to store HVs and 3-input XOR gates for each bit in the resulting HV. Consequently, each temporal encoder is only about 8400 instances, which is a small fraction of the total. The associative memory block is similarly small, with only 1 set of XOR gates and logic to return the popcount for one HV per class. In this work's implementation, the associative memory is not stored in SRAMs because of the relatively small number of classes we compare against. The Hamming distance calculation is a design space worth exploring for tasks with much larger numbers of classes, so that classification latency and power/area can be traded off with each other. Finally, the spatial encoder dominates the remaining area and power. This is expected because its size scales with the number of channels within each modality, and within each ones there are muxes, XOR gates, and counters used to perform the bundling and binding operations needed for spatial encoding. Spatial encoding is already being performed sequentially across all the modalities of a channel, so one way to reduce its footprint further would be to have a single spatial encoder over all modalities, adding more latency.

4.5 ASIC Discussion

Based on the synthesis results, there is potential for much more hardware optimization. The first problem to tackle is reducing the size of the item and feature projection HV memories. As alluded to in Chapter 4.4, the memories are essentially just a collection of random HVs. Since the only requirement to maintain orthogonality is that each channel across all modalities needs to use different HVs, one could devise a method where a common collection of HVs is used but the rows of that collection is randomly chosen before being sent to each modality's spatial encoder item & feature projection HV inputs. Since each channel within a modality is being calculated sequentially, access to a different random row of the common memory for every subsequent channel is all that would be needed. This was validated in Matlab using a common set of 105 HVs, with the feature projection HVs accessed using the `randperm(105,1)` function and common item memories for all modalities. The results in Table 4.6 show that the algorithm, even with this scheme, continues to perform better than the best known alternative classification method.

Mean accuracy	Early Fusion	Late Fusion
(1) Synthesized Architecture	78.18%	79.24%
(2) iM & projM rand. of 105	78.97%	76.32%
(3) Best known alt.	74.25%	74.25%

Table 4.6. Accuracy performance of HDC early and late fusion for (1) the synthesized architecture, (2) memory reduction through selecting iM and feature projection vectors randomly out of 105, (3) the best known alternative classification method (XGB).

Since this selection is random, different iterations of this optimization lead to variations in accuracy up to 4%. An additional area for exploration could include running a similar analysis for different HV dimensions, specifically ones larger than 2,000 for which the accuracy variation may decrease due to increased orthogonality. Hardware implementation of the above in order to feed 3 spatial encoders simultaneously would require that the set of 105 HVs be broken down into 4 banks of memory, each of which has 2 ports. This is because over 3 modalities, generatation of 6 distinct feature projection HVs and 1 common item HV is needed. There would need to be one pseudorandom number generator in the system with a fixed seed that must be common to training and inference, and functions that will generate distinct memory bank and row addresses for for all 7 desired HVs. The output HVs from these banks will be multiplexed and routed to their appropriate destination based on which bank it was accessed from. This logic overhead would be very minimal in contrast to a further 3x saving in memory area compared to this work's implementation, resulting in a memory area contribution of only 25%.

There may even be a possibility to compress this even further, as there only needs to be 105 distinct *combinations* of HVs for feature projection vectors and item memories across all channels. Given that $\binom{14}{3}/\binom{3}{2} = 121.3$, this could potentially mean we would only need 14(!) HVs if there was a method to generate all the distinct combinations. More interesting is the potential of extending this optimization to applications with larger numbers of sensors. The effect of this optimization on other datasets would help explore different types of classifications that may or may not be as resilient to replication across modalities. To what extent can this technique still maintain sufficient performance? Could the number of modalities be artificially increased in order to take advantage of this discovery? If the iM & feature projection HVs are able to be shared across modalities, increase in modalities would decrease the channel count of the largest modality and increase the re-use of the encoding HVs.

The usage of ROMs instead of SRAMs would further improve the memory area by around 30-40% and the memory power by around 20% based on the comparison shown in Table 4.5, though with 30% increase in leakage power for a 32×125 ROM size and 33% increase in cycle times, there may be a point at which this no longer becomes beneficial. This is especially true if only $\sim$14 HVs are used, depending on the ratios between dynamic power and leakage power for both SRAM and ROM of that memory size. Additionally, given the potential order of magnitude compression of iM size, efforts may be better spent slimming down the logic in the spatial encoders, specifically the bundling operation that performs bitwise addition. It is not clear what can be done about the D number of $log_2(channels)$-bit counters. More efficient popcounts could be used, but only if multiple channels were processed simultaneously, which has even larger area penalty. Finally, the training state machine and feature extractors need to be added to the RTL implementation to also explore the training process. Since training uses the same underlying datapath, the above observations still apply, but it remains to be seen what the impact of the feature extractors would be. Furthermore, feature extraction blocks are different between different types of sensors - another direction for future research.

Chapter 5

Vectorized HDC Sensor Fusion

5.1 Implementation

5.1.1 Algorithm optimization

Another area of interest that is explored in this work is the vectorizability of the HDC algorithm. The basic binary operations used in HDC applied to very-large vectors suggest that vectorization could provide significant speedups. For very-large number of channels such as in the sensor fusion case, optimization of the spatial encoder is of particular interest since it dominates the overall operation count. Targeting the open-source Rocketchip RISC-V CPU, preliminary optimizations of the early fusion sensor fusion algorithm architecture were made. This, based on the success of this method for ASIC memory storage, included using the same iM for each modality. The optimization of using a multiplexer for the feature projection vectors was also preserved. However, a dimension of $D = 10,000$ was selected to fully explore the limiting blocks at full capacity instead of reducing down. The result of the multiplexed feature projection is shown in Figure 5.1 to still perform better than the best alternative classification method for $N = 3$ and $D = 10,000$.

5.1.2 Software: C

The HDC multi-modal sensor fusion algorithm was then mapped from MATLAB to the existing C code from [5]. First, the code was changed to remove the optimizations for the PULP platform and to target the RV64 ISA by using 64-bit instead of 32-bit unsigned

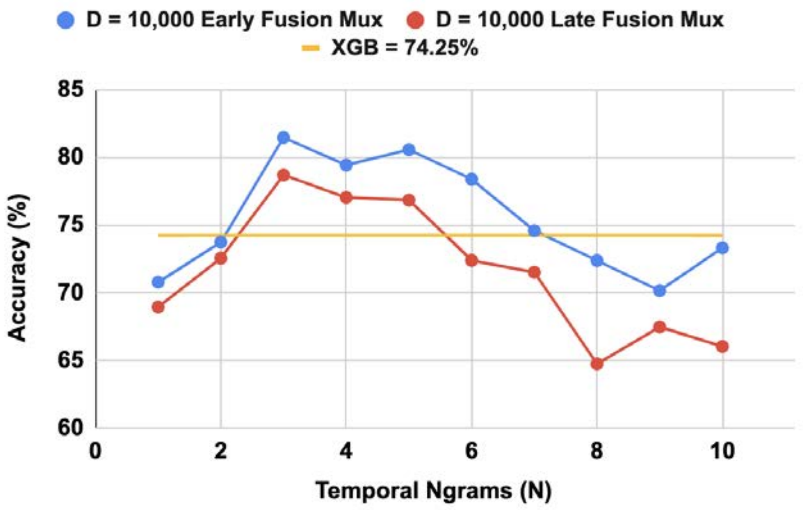


Figure 5.1. Mean accuracy using D = 10,000 and multiplexed feature projection vectors for early and late fusion compared to the best known alternative (XGB) vs. temporal Ngrams

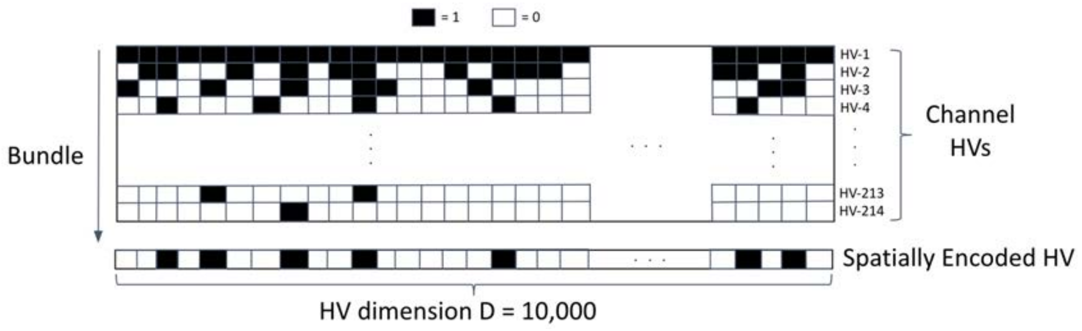


Figure 5.2. Visualization of 214 channel features encoded into HVs to bundle in spatial encoder

integers, which provides a theoretical 2× speedup. The bit-manipulation extensions to the RISC-V ISA [15] were not retained because the Rocket core does not yet support this draft specification, and no multi-core optimizations were implemented because the target platform was Hwacha instead of a multi-core Rocket. Next, the architecture was modified to accommodate the multi-modality dataset, which triples the number of spatial encoders—a large penalty since it is the most computationally-intensive block. Finally, much effort was put in to optimize the code in order to be able to vectorize it efficiently later, as described below. Some of these optimizations that actually changed the algorithm were re-validated in MATLAB to ensure that they maintained accuracy by not affecting the orthogonality of HVs.

Listing 5.1. PULP Naive Mode

```
//componentwise majority: extract the value of the ith bit of
//each row into "majority"
//then compute the popcount w/ numOfSetBits(uint64_t i)
//for bundled query HV
uint64_t majority = 0;
for(int z = 63; z >= 0; z--){
    for(int j = 0 ; j < channels + 1; j++){
        majority = majority | (((chHV[j][i] & (1<<z)) >> z) << j);
    }
    if (numOfSetBits(majority) > channels/2) {
        query[i] = query[i] | (1 << z);
    }
    majority=0;
}
```

Optimization 1: Spatial Encoder

The first major optimization was made to the spatial encoder block, which has by far the most instructions executed given the large number of channels that need to be processed. The most computationally-expensive operation here is finding the mode across all channel HVs, which is visualized in Fig. 5.2 for 214 channels of dimension $D = 10,000$. To count the number of black squares in each column, a word-parallel majority count (popcount) must be performed. The PULP-HD implementation is naive, looping through every single 32-bit word, extracting each bit into a new unsigned integer and then executing a popcount on that, as shown in Listing 5.1. This is highly inefficient due to the 3 nested loops and excess memory accesses to the large `chHV` (channel HV) variable, but was sufficient for their implementation due to a small number of channels (4).

An intermediate optimization for sufficiently large number of channels is to break the HVs up into a set of bit matrices (i.e. 64-element array of 64-bit unsigned integers), transposing them, and then performing a popcount on each row, as visualized in Fig. 5.3. This

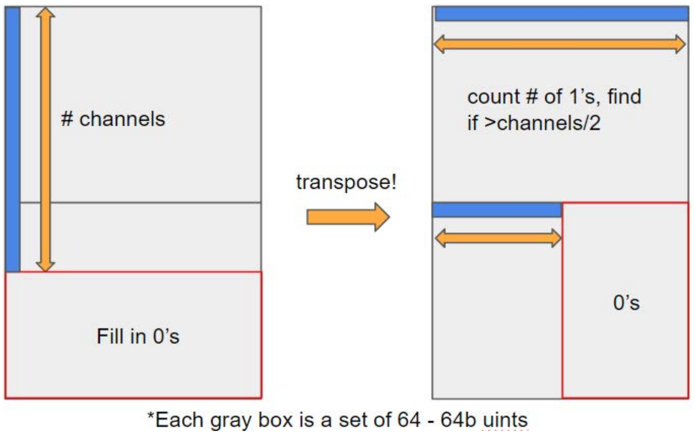

# channels
Fill in 0's
transpose!
count # of 1's, find if >channels/2
0's
*Each gray box is a set of 64 - 64b uints

Figure 5.3. Bit matrix transpose and popcount

Listing 5.2. Most efficient RV64 popcount

```
int numberOfSetBits(uint64_t i){
    i = i - ((i >> 1) & 0x5555555555555555);
    i = (i & 0x3333333333333333) + ((i>>2) & 0x3333333333333333);
    return (((i+(i>>4))&0x0F0F0F0F0F0F0F0F)*0x0101010101010101)>>56;
}
```

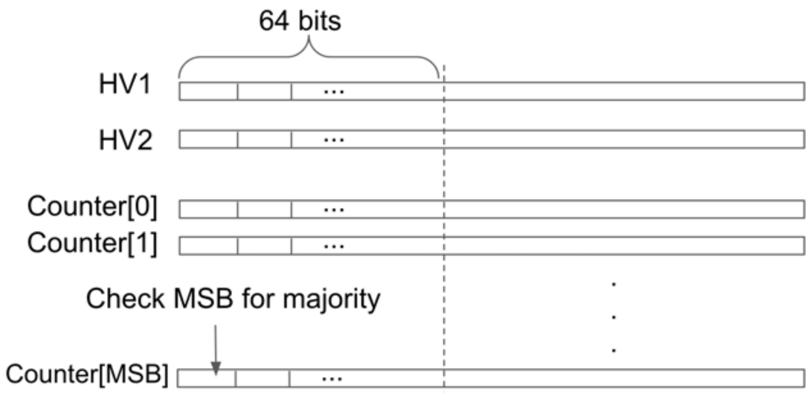


Figure 5.4. Bit serial, word parallel popcount

was implemented using an algorithm from [18]. However, while this removes 1 loop, it still requires *channels* number of unsigned integers to be concurrently stored in memory, and cannot benefit from a single-instruction hardware popcount because it is not yet implemented in the RISC-V ISA. Instead, the most efficient popcount instruction using RISC-V instructions is shown in Listing 5.2.

The final optimization was to recognize that there was actually no need to do any transposition. Instead, using a trick similar to the transition counting implemented in [19], separate unsigned integers were used to represent every bit of 64 parallel counters. The resulting query HV is just the MSB word of these counters, as shown in Fig. 5.4, provided that the counters are initialized properly given the number of channels. This allows for the `chHV` variable to be implemented as just a single unsigned integer by performing the channel HV mapping and majority update in the same loop in a bit-serial, word-parallel fashion. Using this method, the memory footprint in the spatial encoder stays constant as channels increase and vastly improves its vectorizability. The code for this final implementation is in Appendix A (note: some intricacies in the actual implementation are omitted for clarity).

Optimization 2: Spatial Encoding Ordering

The next optimization was to the ordering of the spatial encoding blocks in the main method, afforded by the memory savings from above. The PULP-HD implementation calculates N query HVs to feed the temporal encoder at every input sample, which means that $N \times \textit{modalities}$ spatial encoders are run, instead of having a sliding window of N precomputed query HVs. Implementing this does require N HVs to be stored concurrently, a relatively small penalty, but drastically reduces the amount of cycles devoted to spatial encoding by a factor of N.

Optimization 3: Temporal Encoder

The permutation operation needed for temporal encoding, which is nominally a 1-bit cyclical right shift, is quite expensive as shown in Listing 5.3 from the PULP-HD implementation. It requires figuring out the overflow bit from every 32-bit unsigned integer in a ripple-shifting scheme. To speed up this operation, it was noted that the point of permutation is to remove correlation between HVs in every binding iteration of the temporal encoder. Therefore, two CPU-friendly optimizations were explored: 1) logical right shifting

Listing 5.3. PULP-HD 1-bit cyclic shift

```
//Here the hypervector q is shifted by 1 position as permutation,
//before performing the componentwise XOR operation
overflow = q[0] & mask;
for(int i = 1; i < bit_dim; i++){
    old_overflow = overflow;
    overflow = q[i] & mask;
    q[i] = (q[i] >> 1) | (old_overflow << (32 - 1));
    q[i] = q_N[i] ^ q[i];
}
old_overflow = overflow;
overflow = (q[bit_dim] >> 16) & mask;
q[bit_dim] = (q[bit_dim] >> 1) | (old_overflow << (32 - 1));
q[bit_dim] = q_N[bit_dim] ^ q[bit_dim];
q[0] = (q[0] >> 1) | (overflow << (32 - 1));
q[0] = q_N[0] ^ q[0];
```

by 64 bits, thereby injecting 64 0's and only requiring an index pointer increment; and 2) logical right shifting by 1 every 64 bits, thereby injecting 1 zero every 64 bits. Both approaches were validated in MATLAB to show that accuracy is hardly impacted using either scheme. In the end, approach two was taken, as this is more easily vectorizable, despite requiring more instructions to execute on CPU.

Optimization 4: Memory Footprint

Finally, memory optimizations were considered from the algorithm side. Though this particular algorithm can tolerate HV dimensions down to 2,000, 10,000-dim HVs were retained because it is a commonly-used size to balance HV size against orthogonality and to explore the full-scale vectorization implications. It was previously noted that the spatial encoding takes an item memory HV and 2 feature projection HVs to embed the information from each channel. Since these HVs are nominally all orthogonal, memory usage could be saved by sharing the item memory across all 3 modalities as done for the HDC sensor fusion ASIC, resulting in a 2x item memory footprint reduction. Additionally, bundling the outputs from all the channels across all modalities at once (instead of 2-step: within each modality, then across modalities) was considered due to reduced computation with the transpose popcount scheme, but because the number of channels per modality differs, this places excess weight on modalities with more channels and degrades accuracy for this dataset.

Finally, randomly assigning feature projection HVs to each modality from a common bank of HVs was also explored after the technique was discovered during ASIC optimization attempts. However, a pseudo-random number generator in the C code that could match what was used in the MATLAB training to be able to evaluate the memory footprint reduction was not implemented; it has been left for future work. In theory, as long as each channel within a modality receives an orthogonal combination of item memory and feature projection HVs, a common bank of HVs could be indexed into with the absolute minimum of HVs needed a result of the combinatorics problem to get 105 distinct combinations (limited by EEG channel features). The critical piece here would be the method to generate all the combinations using only 14 HVs—also left for future work.

5.1.3 Software: Hwacha

Targeting Hwacha [20], the vector unit developed at UC Berkeley, requires that vector assembly code be written inline with the original C code. The approach was to break up the datapath into blocks that could maximally utilize vector registers in place of memory loads/stores, and then vectorize those blocks. The first block that was vectorized was the 3-input majority function. This implements the `(A&B)|(B&C)|(C&A)` operation for the query vectors after spatial encoding for each of the 3 modalities. In the code repository, this can be found in the `vec_majority_3_asm.S` file.

The second block that was vectorized was the spatial encoder. The optimizations described above were crucial to being able to use a minimal register footprint in the vector unit and minimize loads/stores. In total, only the number of majority counter bits + 4 data registers, 4 address registers, and 5 shared registers needed to be used across all the vector fetch blocks. Manual unrolling of some loops was required for maximum register reuse, and versions for a 6-bit and 7-bit majority counter were written for the GSR and ECG/EEG modalities, respectively due to the differing numbers of channels. There are only 2 vector loads required for each channel and 1 vector store at the end of spatial encoding. In the code repository, this can be found in the `vec_computeNgram.c` and `vec_computeNgram_asm.S` files.

The third block that was vectorized was the temporal encoder. As described above, the final implementation was the permutation with logical right shift followed by the binding operator. This is executed for each of the N HVs generated by the spatial encoder. Since the N parameter was selected to be three during the algorithm design section for optimal performance, this loop was unrolled to reduce the unnecessary additional loading and storing of the output at each iteration. The final code mapped to just three loads, one stores, two shifts, and two XORs. In the code repository, this can be found in the `main_vec.c` and `ngram_bind_asm.S` files.

The final block that was vectorized was the associative memory. The number of classes was unrolled and hardcoded for maximum efficiency and stored in the vector data registers. Part of the Hamming distance is calculated using just XOR and the vector equivalents of the popcount from Listing 5.2. However, the accumulation of the popcounts between the 157 unsigned integer elements in the vector could not be implemented on Hwacha due to the inexistence of a vector accumulation instruction, and is instead done on the Rocket core. Since there are only two classes to compare, this penalty is not important compared to the spatial encoder. In the code repository, this can be found in the `fusion_funcs.c`, `vec_hamming_distance.c`, `associative_memory_vec.c`, and `AM_v.S` files.

5.2 Vectorized HDC Evaluation

The C code and inline Hwacha assembly were simulated at the block- and top-levels using Spike (with `--extension=hwacha`) and in VCS on the default `HwachaRocketConfig` found in Chipyard [17]. This config has 1 large Rocket core, 1 default-sized Hwacha lane, and an 8-way 512KB L2$. For each optimization and block, the number of cycles needed was compared pre- and post-optimization and between the Rocket-only and Hwacha implementations. It is important to note that Spike is unable to count the number instructions executed on Hwacha, so speedup numbers on the vectorized code is only given from the VCS simulations. Additionally, the instruction used to print the number of cycles actually consumes some 10's of instructions, so the actual speedups may actually be slightly higher than reported.

Since the task implemented was inference-only, functional correctness was verified by comparing classification results against the golden MATLAB model. Each optimization that altered the algorithm (e.g. the new permutation algorithm) would change the golden accuracy, and since HDC is fundamentally somewhat random, these accuracy numbers would vary within a couple percent. Unfortunately, the full inference task was not able to be run on VCS due to the sheer number of cycles required to get through all 380 input samples, so accuracy was verified in Spike and overall speedup numbers are generated using only the first 30 input samples for Rocket-only vs. Rocket with Hwacha and the first 10 input samples for comparison with the original unoptimized work.

5.3 Vectorized HDC Results

5.3.1 CPU-only Optimizations

The first optimization, which was the transposition of the bit matrices, resulted in a **6.8**× reduction in the number of cycles required for the 32-channel GSR modality as compared to a 64-bit version of the original PULP-HD implementation. It is important to note that the transpose method has a number of instructions that grows by $ceil(channels/64)$ as opposed to *channels* from the original method.

By moving to the bit-serial, word parallel popcount method, an additional **4.64**× speedup was achieved for the 32-channel GSR modality. Unfortunately, this speedup decreases as the number of channels increases vs. the transpose method because it too grows proportional to *channels*. Given that ECG (77 channels) and EEG (105 channels) were sped up by 3.23× and 2.39×, respectively, it appears that the breakeven point will be somewhere around 250 channels per modality. This number would be approx. 2.3x lower if the RISC-V bit-manipulation instructions could be used for the transpose method, based on the Wolf speedup numbers in [5]. Furthermore, by combining the channel HV mapping and popcount instructions into the same inner loop, the memory footprint was reduced, resulting in **3.3 - 8.7%** reduction in cycle count in VCS simulations (higher with more channels), suggesting there were fewer indices to calculate and increased cache hits.

By ordering the spatial encoders into a sliding window, an exactly **3**× speedup was achieved, which is expected because the spatial encoders dominate the overall cycle count and because of the selected $N = 3$. This benefit would undoubtedly increase with larger N, which may be needed to maximize accuracy for other classification tasks. Based on this result, the extra memory requirements needed to store N query HVs seems to have been far outweighed by the number of cycles required for the spatial encoding computation itself.

By moving to the logical right shift by 1 for every unsigned integer, a speedup of just **11.8%** is seen in the temporal encoding. Further investigation shows that this operation is actually memory bound, as it needs to load from and store into the query HVs in memory, which may be in disjoint places after the initial spatial encoding. This is confirmed by the Spike simulations, which suggest that there are just 77.5% fewer instructions required by using the shifting despite having many fewer operations in the C code.

5.3.2 Hwacha Acceleration

In Table 5.1, the speedup from mapping to a single Hwacha lane for each of the major blocks and in aggregate in comparison to the Rocket implementation with all the above CPU optimizations enabled is shown.

Block	Condition	CPU cycles	Hwacha cycles	Speedup
3-input Maj.		2,437	849	2.87×
Spatial Enc.	32 ch.	340,439	26,296	12.95×
Spatial Enc.	105 ch.	1,275,872	95,424	13.37×
Temp. Enc.	N = 3	3,313	58	57.1×
Assoc. Mem.	2 classes	9,879	2,592	3.8×
All	30 samps.	75,844,682	6,083,890	12.47×

Table 5.1. Speedup: Hwacha vs. CPU (post-optimization)

When run on 30 input samples, an overall speedup of **12.47**× is achieved! This is better than the speedup presented in [5] for the most directly-comparable implementation: the 8-core Wolf vs. 1-core without bit manipulation extensions (7.96×). This comparison makes the most sense because the default Hwacha configuration has 8 sequencer entries, which to the first order parallelizes computations 8 ways. Further, the speedup presented in [5] was demonstrated for an application with only four channels as compared to the multi-modal model with 214 total channels used in this work. This shows that the optimizations selected for the spatial encoder are beneficial for even 2 orders of magnitude larger number of channels.

A deeper dive into the blocks reveals that the maximum speedup is obtained on vector fetch blocks that have have a higher proportion of load/store operations (temporal encoder) vs. computation (spatial encoder). This shows that the higher memory interface bandwidth and decoupled access/execute architecture of the Hwacha accelerator benefit the temporal encoder. Still, the ∼13× speedup of the spatial encoder is very significant and, as expected, dominates the overall speedup of the algorithm.

5.3.3 Overall Speedup vs. PULP-HD

To measure the overall speedup of the vectorized implementation versus the the original PULP-HD implementation translated to RV64 on a single Rocket core, some extrapolations had to be made. This is due to the fact that VCS simulations on all input samples using the non-optimized implementation takes a very, very long time (it needs to simulate 10's of billions of cycles). Therefore, an extrapolation is made based on the speedup result from the previous section combined with some Spike simulations, as well as simulations on a small subset of samples (10 total).

Version	Instruction Count
Original PULP-HD	11,728,914,671
CPU optimized	734,919,671
Vectorized w/ Hwacha	15,159,673

Table 5.2. Spike simulation instruction count for full inference task (380 samples)

Spike simulations on the entire inference task (380 samples) yields the instruction counts as shown in Table 5.2. Since Spike is unable to count Hwacha cycles, the last two rows suggest that Spike overcounts the Hwacha speedup by 3.9×. The overall Spike speedup is 773.7×, so the estimated actual speedup should be on the order of 773.7/3.9 = **198**×!

Version	Cycle Count
Original PULP-HD	266,984,854
CPU optimized	25,360,374
Vectorized w/ Hwacha	2,050,319

Table 5.3. VCS simulation cycle count for 10 samples

VCS simulations for 10 samples were successfully run in reasonable time. The VCS cycle counts are shown in Table 5.3. The total speedup is 130.2×. This speedup result is hindered by the cold start penalty from the sliding window temporal encoder optimization, which would have a significant impact on the comparison for a smaller number of inferences—in this case only 7 out of the total 377. The gains should approach the predicted speedup of 198× as the number of inferences increases.

5.4 Vectorized HDC Discussion

A speedup of 198× would have a significant impact on not only sensor fusion applications, but specifically the recall of reactive behavior paradigm that this work proposed as it reduces the total time per classification for each input sample. Reducing the overall computation time for the dataset is generally desirable in most classification tasks, but specifically in prosthetic embedded sensor fusion applications where real-time classification is critical to prosthetic performance, latency between sensor signal samples and decisions for prosthetic action could result in the difference between success or failure of task completion. This is especially true in the reflex action case where prosthetic response times should aim to be in par with human reflex delays (0.15 seconds in reaction to touch stimulus [21]). Additionally, for prostheses, not only do classifications need to continuously run throughout its usage, but the overall usage time is a significant factor in usability. Reduction of cycle counts per classification should also result in reduction of energy per classification, improving usage time for embedded applications. Further work should be done to quantify the energy per classification improvement as a result of these optimizations.

Based on the results of this work, there are two ISA-specific optimizations that would yield additional performance beyond this work: 1) integration of the draft RISC-V bit manipulation ISA extensions [15] which would implement the single-instruction popcount, and 2) implementation of a proposed addition to the Hwacha vector reduction instructions to accumulate the value across a vector register into a shared register, which would help in the Hamming distance calculation.

This work's optimizations did not require additional vector lanes or sequencer entries because each block of computation was able to be performed within one iteration of the `vsetvl` instruction due to this work's efficient reuse of vector data registers. Conceivably, adding vector lanes or sequencer entries would add additional speedup, but it would run into Amdahl's law and soon be constrained by the memory load/store bandwidth.

Extensive benchmarking of cache utilization like in [5] would inform further optimizations that could be made. While the optimizations in this work already drastically reduce the memory footprint, only a handful of HVs already exceed the size of L1$, so it would be interesting to see if variables could be compacted together in such a way to maximize cache hits. Future work should also analyze the impact of parameters like [5] in order to analyze how to maximally use the cache hierarchy and the resulting latency improvement. There is also more work to be done to implement the training in software and to vectorize it. The underlying computations and optimizations here are all reused, but the only additions would be some additional feature encoders and methods that write to the associative memory instead of reading from it.

On the algorithms side, some of the on-the-fly HV generation, approximate majority, and single-cycle Hamming distance schemes employed in [16] appear to be directly applicable to architectures other than FPGAs. Implementing those techniques on this work's algorithm could yield additional speedups in the spatial encoder and associative memory blocks, whilst remaining vectorizable. Another area left for future work would be extending the vectorization to architectures with much larger associative memories and observing the corresponding growth in cycle count—at which point does it become the dominating block?

Finally, these optimizations are broadly applicable to other ISAs with vector extensions, such as Intel's AVX-512. It would be a good test to see if the built-in GCC support for those extensions are able to compile this work's final C code as optimally as the hand-written assembly used for the Hwacha vector accelerator.

Chapter 6

HDC Recall of Reactive Behavior

6.1 Force Sensor Characterization

Towards the goal of integrating force sensors into the EMG classification system developed in [8], Tekscan A201 force sensors shown in Figure 6.1 and Figure 6.2 were selected for replicability and precision. The resistive force sensors are placed in a voltage divider with the selected reference voltage and unity-gain buffer to get an output signal that changes with the force applied. In order to improve the replicability and stability of the signal for weights with larger radii, a bumper was added to centralize the force onto the sensor area.

The force sensors were characterized over time in that setup for weights between 20g and 500g—approximately the weights expected to be used for concept demonstration. The results are shown in Figure 6.4 and Figure 6.3. The removal of force is reflected in the signal within a second, but the addition of a force results in a signal that drifts over time. The differences in the signal between weight amounts is also not very linear, this suggests that for any learning algorithm, differentiating between smaller weights will be more challenging

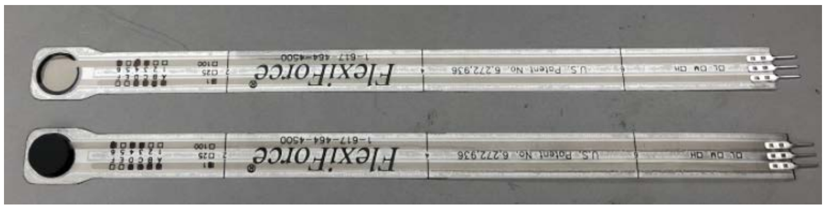


Figure 6.1. Selected force sensor top view, bumper added on bottom sensor

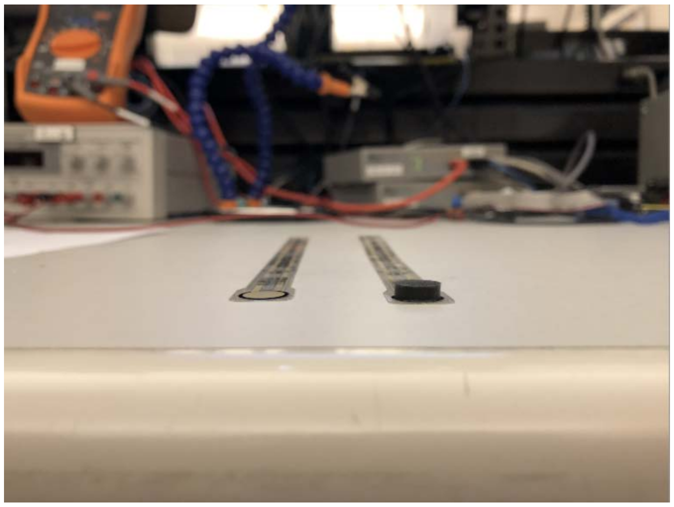


Figure 6.2. Selected force sensor front view, bumper added on right sensor

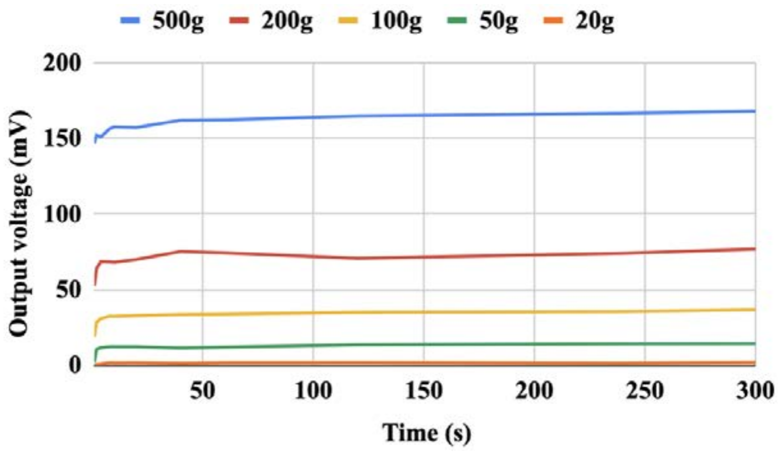


Figure 6.3. Force sensor characterization over time with weight added at t=0s

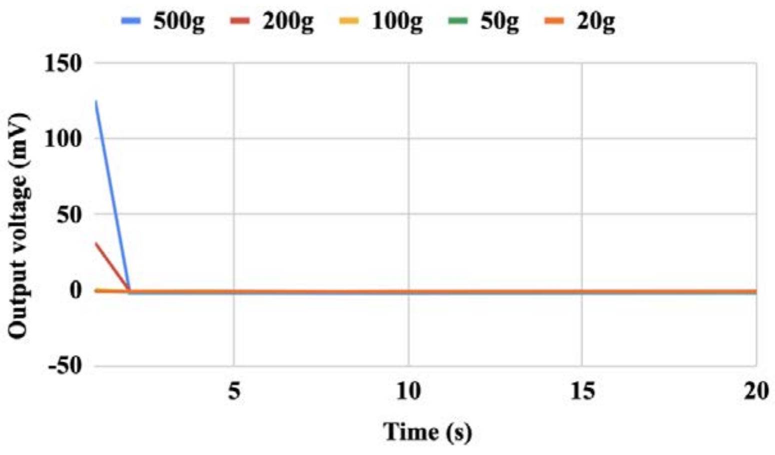


Figure 6.4. Force sensor characterization over time with weight removed at t=0s

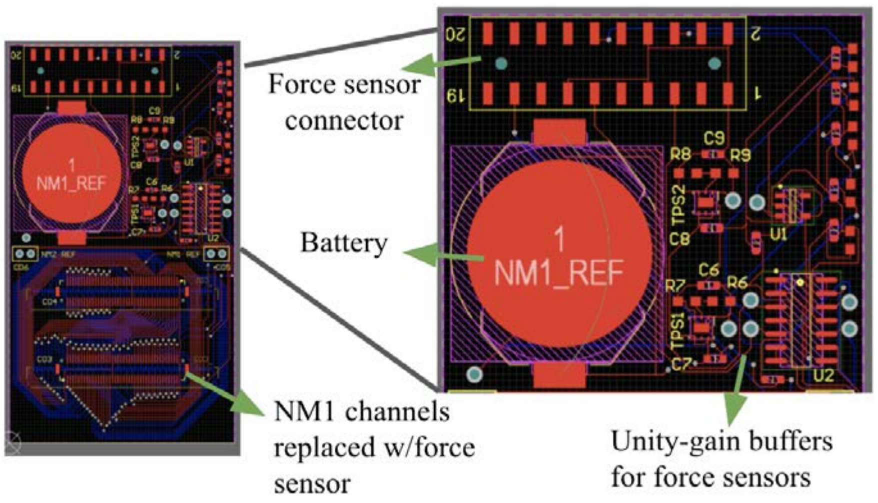

Force sensor connector
Battery
NM1 channels replaced w/force sensor
Unity-gain buffers for force sensors

Figure 6.5. Force-flex-EMG Adapter PCB

than between larger ones. Additionally, the sensor range is quite large so its replicability on smaller force signals is less. This characterization is useful in designing protocols to best demonstrate the HD recall of reactive behavior technique.

6.2 EMG & Force Sensor Integration: PCB

Previous work for the EMG classification system [8] designed a PCB to interface with the flexible EMG sensor arrays. In this work, both force sensors and EMG sensors are integrated into the system. As a result, the original PCB was modified to replace five EMG channels with force sensors. The additions include a force sensor connector, op-amps for unity gain buffers and resistors for the voltage dividers for each sensor, and a 3V coin cell battery to power the voltage supplies and reference voltages. The design is shown in Figure 6.5. It was fabricated and is waiting on testing.

Future work will involve design of experiments using this platform to demonstrate the ability to build a program memory between various sensors and actuators. Exploration of context-based shared control could involve different program memories depending on additional information from orientation and motion sensors. Limitations on the number of states that can be stored into the program memory given that states may be similar to one another, and the ability of the decoder to discriminate between these states of motion will be determined.

Chapter 7

Conclusion & Future Work

In this work, the accuracy and hardware implications of early fusion and late fusion for a multi-modal Hyperdimensional Computing paradigm were explored. In contrast to other traditional ML algorithms, due to the inherent binding of information without exceeding HV capacity, early sensor fusion not only performs as well as late sensor fusion, but actually performs better for certain parameters in HDC. Both versions of the algorithm were implemented in Verilog by modifying a previous codebase to add additional parallel blocks for each sensor modality and fit the temporal parameters selected during the algorithm design process.

Several preliminary optimizations were made in anticipation of reducing the overall ASIC hardware costs once synthesized. These included reducing the HV dimension to 2,000, multiplexing feature projection vectors, sharing iM and feature projection vectors across each modality, and using the same iM and feature projection vectors for both strength of emotion and polarity of emotion classification. These changes reduced the memory size from an initial 642 HVs to 315 HVs (-50.9%).

Final synthesis results demonstrate the hardware implications of early fusion vs. late fusion. Early fusion reduced the total area by 4.3% and the total est. power by 6.2%. The breakdown of area and power by block validates the expected results showing that the temporal encoder increases from 3.1% to 8% of the total power and from 2.0% to 5.8% for area between early to late fusion due to the additional two temporal encoders required for the multiple modalities in late fusion. However, in contrast to the overall demands of the spatial encoder and memory, the effect of removing two temporal encoders is small.

Final synthesis results showed that the overall area due to the memory was 47% for early fusion and 45.1% for late fusion and the overall power due to memory was 49.9% for early fusion and 47% for late fusion. Simple calculations show that without the memory optimizations made, the memories alone would've been 72% for early fusion. This demonstrates the significant impact of hardware-software co-design. The spatial encoder also contributes a significant amount towards the power (48.9% for early fusion and 46.4% for late fusion) and area (42.7% for early fusion and 39.6% for late fusion).

Given these results, it is clear that effort would be better spent trying to further optimize the memory requirements and spatial encoder. In that direction, preliminary algorithm exploration was done to further reduce the size of the iM and number of feature projection vectors. Using only a common set of 105 HVs, the iM is selected sequentially, but the 6 required different feature projection vectors were randomly selected and assigned to the 3 modalities. This resulted in slight performance degradation and varying accuracy on different iterations due to the randomness involved, but HDC still performed better than the next best classification method even when using this method.

This result suggests that there is much room for exploration in this area, specifically for the sensor fusion application where the presence of different modalities allows for further reduction and potentially acceptable correlation. The possibility of further compressing the memory size to only the number of HVs necessary to generate 105 distinct combinations of HVs from feature projection vectors and item memories across all channels could mean that only 14 HVs would need to be stored given a method to generate combinations. At that point, optimization of the spatial encoder would be required to further reduce the hardware costs.

The vectorizability of a multi-modal Hyperdimensional Computing paradigm was also explored. The early fusion HDC paradigm was implemented on top of [5] with architectural changes made to reflect the multi-modality of the algorithm. Several optimizations were made to the prior work in [5] to maximize speedup. These included new bundling and permutation algorithms, memory footprint improvements, and removal of redundant computations. In contrast to traditional ML algorithms, the operations used in HDC are simple and thus easily vectorizable with significant speedup in a RISC ISA.

To demonstrate this, all of the HV operations in the spatial encoder, temporal encoder and associative memory were vectorized through inline assembly instructions using the Hwacha RISC-V vector ISA extensions. The vector fetch blocks were re-designed to minimize the amount of loads/stores over the vector memory interface and to minimize the number of vector registers used so that the amount of vector chaining could be maximized.

Final simulation results paint a true picture of the actual hardware implications of accelerating HDC algorithms with a vector processor versus multi-core platforms. First, vector processors provide parallelism with more optimized memory loads/stores. Second, the vector processor's registers and cache further reduce the need to share memory resources with the control processor. Combined, this results in a better-than-8× speedup that would be suggested given the sequencer depth of 8. Compared to the original implementation in [5], an approximate 198× speedup is calculated with 12.47× from Hwacha vector acceleration and 15.96× from optimizing the architecture.

Given these results, similar to the ASIC results, effort should continue to be spent trying to further optimize the memory footprint and spatial encoder, some of which has been tackled in [16]. Specifically, the growth in storage and computation demand will increase in sensor fusion applications with large numbers of modalities and/or channels. The possibility of further compressing the memory size to only 14 HVs would significantly reduce storage given a method to generate combinations. At that point, aggressive optimization of data reuse to take advantage of caching and further improvements in the spatial encoder would be required to further reduce cycle count and hardware costs. Integration of the draft RISC-V bit manipulation ISA extensions for single-instruction popcount as well the inclusion to the Hwacha vector accelerator of an accumulation instruction across a vector register for Hamming distance calculation could both also be explored for additional performance improvements.

Finally, a platform for integration of force sensors into the previously developed EMG sensor adapter board was developed. It is clear that not only does HDC performs incredibly well, especially compared to alternative algorithms, on sensor fusion datasets because of the ability to encode any input into the hyperdimensional space regardless of sensor complexity, but also the many successful optimization techniques explored in this work demonstrate that the HDC property of pseudo-orthogonality is highly resistant to simplifications and adjustments made for platform-specific performance improvements.

These results suggest that HDC sensor fusion for the recall of reactive behavior technique - where the modalities are force sensors, EMG sensors, and prosthetic actuator - will be able to maintain high accuracy while still allowing for significant optimization between modalities to reduce the hardware memory and logic requirements for embedded training and inference. Additional sensors could be integrated as well to implement context-based shared control, further utilizing the sensor fusion property of HDC. Exploration using the optimization techniques verified in this work for sensor fusion and the platform designed for HDC recall proof-of-concept together to dig deeper into the concept and properties of recalling reactive behavior for prosthetic reflex response is left for future work.

References

[1] K. Z. Zhuang, N. Sommer, V. Mendez, S. Aryan, E. Formento, E. D'Anna, F. Artoni, F. Petrini, G. Granata, G. Cannaviello *et al.*, "Shared human–robot proportional control of a dexterous myoelectric prosthesis," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 400–411, 2019.

[2] G. K. Patel, J. M. Hahne, C. Castellini, D. Farina, and S. Dosen, "Context-dependent adaptation improves robustness of myoelectric control for upper-limb prostheses," *Journal of neural engineering*, vol. 14, no. 5, p. 056016, 2017.

[3] L. E. Osborn, A. Dragomir, J. L. Betthauser, C. L. Hunt, H. H. Nguyen, R. R. Kaliki, and N. V. Thakor, "Prosthesis with neuromorphic multilayered e-dermis perceives touch and pain," *Science robotics*, vol. 3, no. 19, p. eaat3818, 2018.

[4] L. Osborn, R. R. Kaliki, A. B. Soares, and N. V. Thakor, "Neuromimetic event-based detection for closed-loop tactile feedback control of upper limb prostheses," *IEEE transactions on haptics*, vol. 9, no. 2, pp. 196–206, 2016.

[5] F. Montagna, A. Rahimi, S. Benatti, D. Rossi, and L. Benini, "Pulp-hd: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC).* IEEE, 2018, pp. 1–6.

[6] S. Datta, R. A. Antonio, A. R. Ison, and J. M. Rabaey, "A programmable hyper-dimensional processor architecture for human-centric iot," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 439–452, 2019.

[7] P. Neubert, S. Schubert, and P. Protzel, "Learning vector symbolic architectures for reactive robot behaviours," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'16) and the Workshop on Machine Learning Methods for High-Level Cognitive Capabilities in Robotics*, 2016.

[8] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt *et al.*, "An emg gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 2018, pp. 1–5.

[9] A. Pfeuffer and K. Dietmayer, "Optimal sensor data fusion architecture for object detection in adverse weather conditions," in *2018 21st International Conference on Information Fusion (FUSION).* IEEE, 2018, pp. 1–8.

[10] E.-J. Chang, A. Rahimi, L. Benini, and A.-Y. A. Wu, "Hyperdimensional computing-based multimodality emotion recognition with physiological signals," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS).* IEEE, 2019, pp. 137–141.

[11] J. A. Miranda-Correa, M. K. Abadi, N. Sebe, and I. Patras, "Amigos: A dataset for mood, personality and affect research on individuals and groups," *arXiv preprint arXiv:1702.02510*, 2017.

[12] S.-H. Wang, H.-T. Li, E.-J. Chang, and A.-Y. A. Wu, "Entropy-assisted emotion recognition of valence and arousal using xgboost classifier," in *IFIP International Conference on Artificial Intelligence Applications and Innovations.* Springer, 2018, pp. 249–260.

[13] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE International Conference on Rebooting Computing (ICRC).* IEEE, 2017, pp. 1–8.

[14] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *2015 IEEE Hot Chips 27 Symposium (HCS).* IEEE, 2015, pp. 1–39.

[15] B. Koppelmann, P. Adelt, W. Mueller, and C. Scheytt, "Risc-v extensions for bit manipulation instructions," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS).* IEEE, 2019, pp. 41–48.

[16] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 4, pp. 1–25, 2019.

[17] Berkeley-Architecture-Research, "Chipyard framework," https://github.com/ucb-bar/chipyard, 2020.

[18] H. S. Warren, *Hacker's delight.* Pearson Education, 2013.

[19] R. Krashinsky, S. Heo, M. Zhang, and K. Asanovic, “Sychosys: Compiled energy-performance cycle simulation,” in *Workshop on Complexity-Effective Design, 27th International Symposium on Computer Architecture*, 2000.

[20] D. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar, and K. Asanovic, “Vector processors for energy-efficient embedded systems,” in *Proceedings of the Third ACM International Workshop on Many-core Embedded Systems*, 2016, pp. 10–16.

[21] E. S. Robinson, “Work of the integrated organism.” 1934.

[22] H. Liew and A. Menon, “cs252_ee290_project,” https://github.com/alisha-menon/cs252_ee290_project, 2020.

Appendix A

Optimized Spatial Encoder

Listing A.1. Optimized Spatial Encoder

```
void computeNgram(int channels, int cntr_bits, float buffer[], \
        uint64_t iM[][bit_dim], uint64_t projM_pos [][bit_dim], \
        uint64_t projM_neg[][bit_dim], uint64_t query[bit_dim]) {
    int cntr_init = (1 << (cntr_bits -1)) - channels/2 - 1;
    uint64_t cntr[cntr_bits], uint64_t temp, carry, chHV;
    for(int i = 0; i < bit_dim; i++){
       //initialize counter to 2^(cntr_bits)/2 - channels/2
       //Counter msb uint becomes the final query HV
       for(int n = 0; n < cntr_bits; n++) {
           if (((cntr_init & (1 << n)) == 0) {cntr[n] = 0;}
           else {cntr[n] = 0xFFFFFFFFFFFFFFFFULL}
       }
        for(int j = 0; j < channels; j++) {
            // calculate channel HV to consume immediately
            if (iM[j][i]^(buffer[j] >= 0.0) {chHV=projM_pos[j][i];}
            else {chHV = projM_neg[j][i]);}
            // incremental popcount
            carry = cntr[0] & chHV;
            cntr[0] ^= chHV;
            for(int n = 1; n < cntr_bits; n++) {
                temp = cntr[n];
                cntr[n] ^= carry;
                carry &= temp;
            }
       }
       query[i] = cntr[cntr_bits -1];
    }
}
```

Appendix B

Code Repository

In the code repository [22], the MATLAB algorithm is contained in the `MATLAB HD Sensor Fusion` folder. The Verilog RTL is in the `Verilog HD Sensor Fusion` folder, the Chisel wrapper is in `generators/src/main/scala/Fusion.scala` and synthesis configurations are in the `vlsi` folder. The C and assembly code is found in the `generators/HD_sensor_fusion/software/fusion` folder.