TECHNION – ISRAEL INSTITUTE OF TECHNOLOGY
FACULTY OF ELECTRICAL AND COMPUTER ENGINEERING

VLSI LAB

PROJECT BOOK

SCALABLE HARDWARE ACCELERATOR FOR PERSONALIZED STRESS DETECTION

Students
Hanin Sussan - 207187519
Fawzi Daoud - 209516772

Supervisor
Shahar Gino

1 Introduction

1.1 KNN algorithm:

KNN, which stands for K-nearest neighbors, is an algorithm which is widely used in machine learning, and it falls under the supervised learning and classification/regression algorithm category. A supervised learning machine is taught by example, it doesn't make any assumptions about the underlying data distribution and uses proximity to make classifications or predictions about the grouping of an individual data point.

The main goal of the KNN algorithm is to identify the nearest neighbors (K) of a given query point, so that we can assign a class label to that point. In order to do that some parameters should be well defined: first, the distance of one query point to its neighbors that sets the decisions boundaries, meaning the distance that makes a point part of another group hence creating the needed partitions, the most common distance used is called the "Euclidean distance" which is used only on real-valued vectors. Secondly, we need to define the 'K' on which the algorithm will act on. The choice of 'K' is mainly depended on the data that the algorithm will work on; if the data is noisy or has a lot of outliers then it's better to use larger 'K' values. In general, it is recommended to use odd numbers for 'K' in order to avoid ties in classifications.

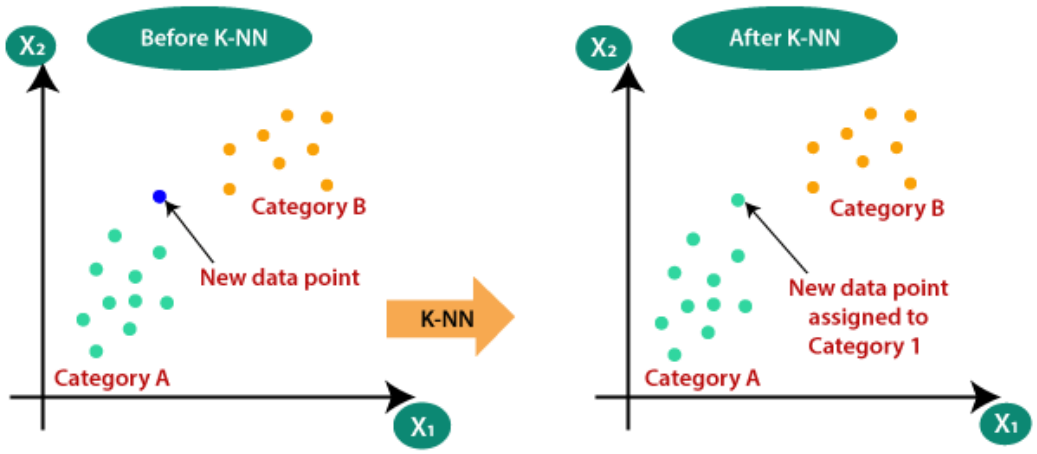
The KNN algorithm is used in many applications, such as: data processing, recommendation engines in web browsing, assessing loan risks in banks, healthcare, and pattern recognition.

Some of the key advantages of the KNN algorithm is that it is fairly easy to implement, it adapts easily when new training data is added and can be sufficiently accurate in many scenarios. In addition, a small number of parameters is needed to be set for it to function (the 'K' and the distance, as mentioned above).

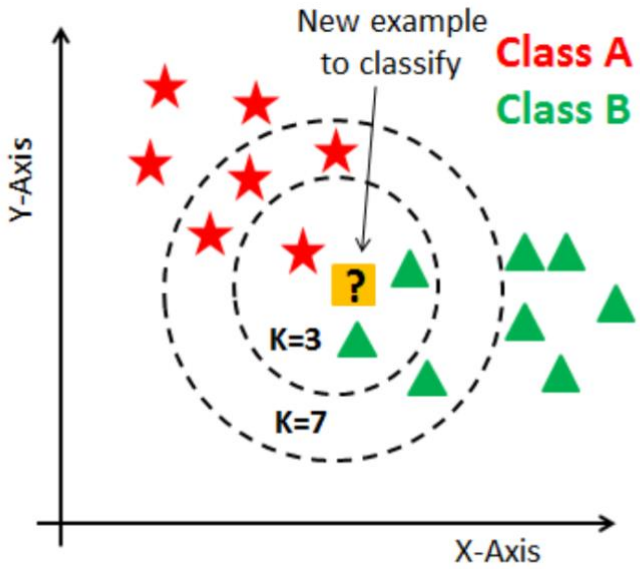
However, the KNN algorithms suffers from few limitations: it doesn't scale well because it uses a lot of memory and storage space compared to other classifiers, meaning it can become very costly to use. In addition, it doesn't perform well with high dimensional data inputs and lastly it is prone to overfit or underfit the data according to the chosen (according to the choice of 'K').

Despite all the mentioned limitations, the KNN algorithms remains popular and fairly effective in many use cases and various domains.

A simple illustration that shows how the algorithm works :



The effect of choosing different values of ‘K’ :



1.2 SVM algorithm:

Support Vector Machines is one of the nonlinear supervised machine learning models, it is a powerful and widely used algorithm for classification and regression analysis, two of the most fundamental tasks in machine learning.

SVM is a type of binary linear classifier that performs the required task, given a set of labeled training data, by finding the optimal hyperplane, a flat subspace of dimension one less the ambient space it is contained in, that separates data points into two classes. The best hyperplane is the one that maximizes the distance from it to the closest data points from each class, which represents the largest margin (separation) between the two classes.

Assuming that the samples are linearly separable and there are no outliers in the data, the hyperplane that lies halfway between two parallel hyperplanes that separate the two classes of data, is the maximum margin. These hyperplanes can be written by SVM formulation that is called Hard Margin SVM.

In cases where the data are not linearly separable or include noise and outliers in the dataset, the Hard Margin SVM simply can't tolerate them, and fails to find the optimization. Therefore, SVM uses kernel tricks to transform the data into a higher dimensional space where it can be linearly separable.

Another solution to the problem mentioned before, is Soft Margin technique. The main purpose of this strategy is to find the hyperplane while allowing an amount of misclassifications or overlaps between classes, which is controlled by a penalty term that is added to the objective function.

Unlike some other models, in which all data points influence the final optimization, for SVM only the support vectors have an impact on the final decision boundary, which determines the hyperplane. Support vectors are the data points that lie closest to the hyperplane; accordingly, they are the most difficult to classify.

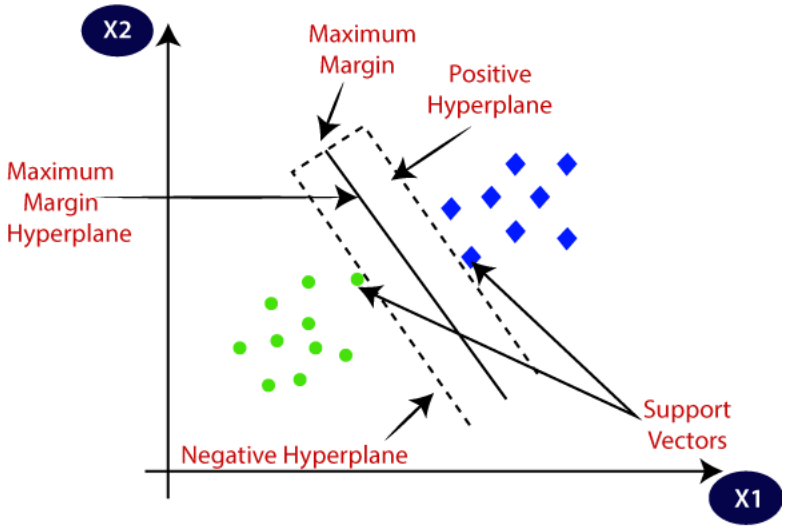
SVMs can be used to solve various real-world problems, such as, text and hypertext categorization, classification of images, classification of satellite data like SAR (Synthetic-aperture radar), hand-written characters recognition and many others in the multiple science fields.

SVMs have several advantages, including their ability to handle high-dimensional data and non-linearly separable data through kernel tricks. Additionally, they are less prone to overfitting compared to other machine learning algorithms, which is important when dealing with limited

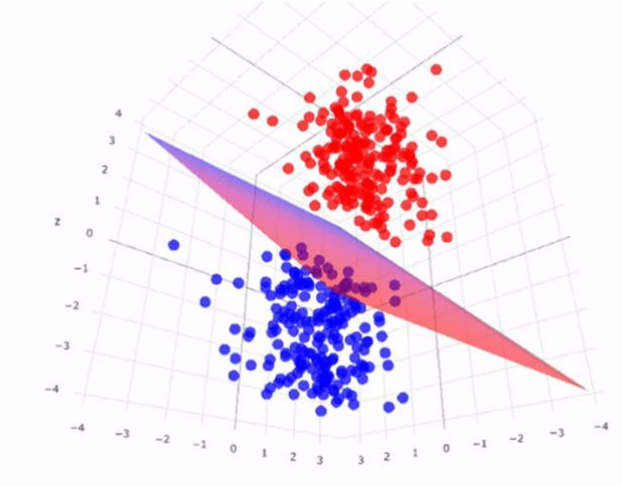
training data. On the other hand, it can be sensitive to the choice of hyperparameters, in addition to the high computational complexity for large datasets.

Notwithstanding all the drawbacks mentioned above, the SVM algorithm remains beneficial in various fields, due to being versatile and effective in a wide range of applications.

An example of a hyperplane between two sets of data points in a 2D space:



An example of a hyperplane between two sets of data points in a 3D space:



1.3 AMBA APB Protocol:

The AMBA APB (Advanced Peripheral Bus) protocol is part of the AMBA (Advanced Microcontroller Bus Architecture) family of protocols developed by ARM. It is a widely used communication interface in digital systems design, particularly for connecting low-speed peripherals to microprocessors or microcontrollers. It is designed based on a reusable based methodology for system on chip (SOC) which is essential in order to meet the current VLSI challenges. It meets those challenges by being low bandwidth, low cost, and minimal power consuming.

The APB interface is not pipelined and is a simple, synchronous protocol that supports both synchronous and asynchronous data transfers, providing flexibility to accommodate different system requirements. Every transfer takes at least two cycles to complete as it works in an FSM model that has three possible states: *IDLE* (the default state), when a transfer is required, the interface moves to the *SETUP* state for a one clock cycle, and it always moves to the *ACCESS* state in the next clock cycle.

It operates as a Master-Slave model where the master is usually a microprocessor (initiates transactions), and the slave or slaves are peripheral devices (respond to the master's transactions).

The communication between the two end points (master and slave) is done using signals that are distributed to the peripheral devices using the APB. The APB interface has many inputs and outputs, and their usage varies according to the accessed peripheral, here are some of the most important signals:

- PCLK: clock signal, all APB signals are time against the rising edge of PCLK.
- PADDR: the APB address bus, can be up to 32 bits.
- PSELx: the Master generates a PSELx signal (in the *SETUP* state) for each slave it intends to communicate with.
- PENABLE: indicates the second and subsequent cycle of the transfer (*ACCESS* state).
- PREADY: used to extend an APB transfer by the slave.
- PSLVERR: used to indicate an error condition on an APB transfer which can occur in both read or write transactions. It is only considered valid during the last cycle of an APB transfer and when PSEL, PENABLE, PREADY are all HIGH.

Example for Read/Write Signal:

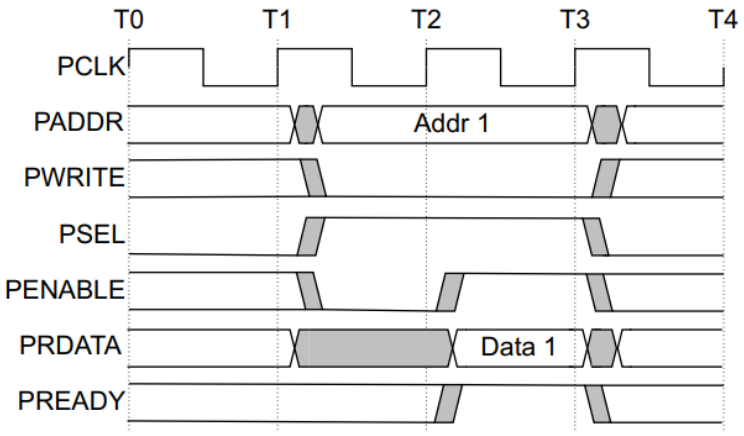


Figure 3-4 Read transfer with no wait states

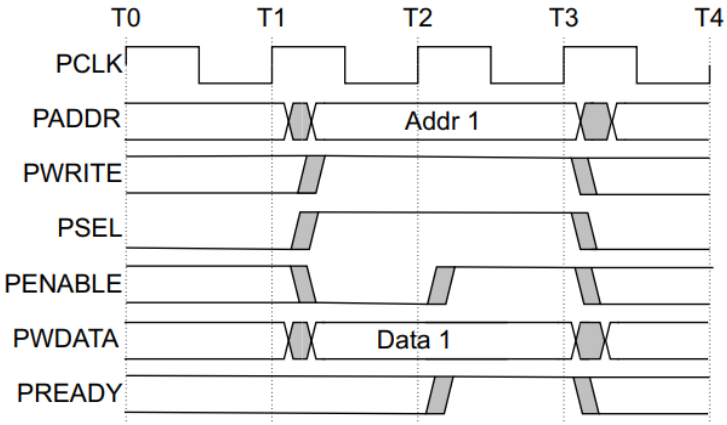


Figure 3-1 Write transfer with no wait states

1.4 Research Paper

Embedded Low-Power Processor for Personalized Stress Detection:

Nasrin Attaran, Abhilash Puranik, Justin Brooks, and Tinoosh Mohsenin

This paper has been enlisted as a fundamental resource for this project, and it is used as a guideline for achieving the project's goal; implementing the proposed processors used to detect stress as a part of wearable technology.

In the prospering field of wearable technology, one of the most important categories is the wearable biomedical systems. Such systems involve processing multiple streams of physiological signals to extract useful information that require a significant number of digital signal processing and machine learning kernels. However, since these systems are wearable, they need to be low powered, lightweight, and capable of real-time processing.

Stress detection is one application of said biomedical systems that tries to monitor and detect in real time changes in the user's body and infer the level of the stress . It achieves this using multi-modal physiological signals, feature extraction techniques, and machine learning classifiers (SVM,KNN). It's done in three important stages: (1) capture and digitize physiological signals; (2) preprocess signals to select/extract features to make intelligent use of the data; (3) transmit relevant information to the user.

The dataset used in this paper is collected during a naturalistic shooting task performed by 15 subjects, in which three levels of stress were induced by manipulating performance feedback: low, medium, and high. The dataset consists of multi-physiological and behavioral recordings and can be used for stress detection evaluation. The paper focuses on studying the low-stress and high-stress conditions.

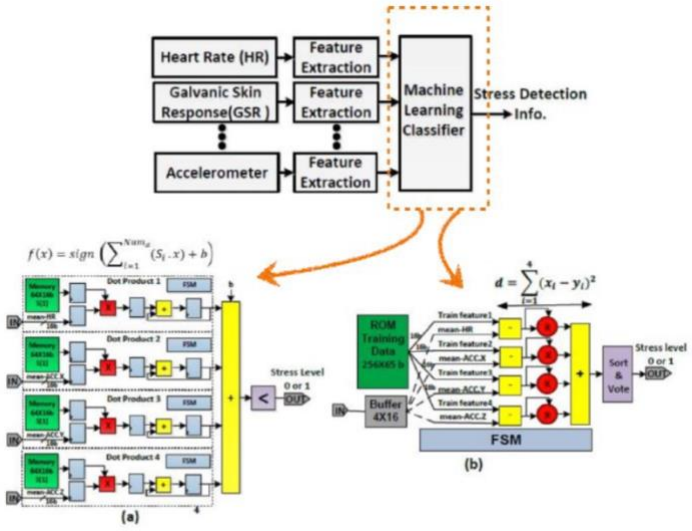
As mentioned before, one major requirement of the system is for it be low powered, which means feature extractions is needed because it allows for a reduction in the number of resources that are necessary to describe a large set of data. In total, 16 features were derived and examined in order to determine which of them contains the most useful information and remove those that don't improve the model's accuracy. The selection was done using the accuracy results of the SVM and KNN classifiers. The final feature set has four features including mean HR, mean Acc.X, mean Acc.Y, and mean Acc.Z, as the best feature set for stress detection. (Acc. Stands for accelerometer). The Scalable and Pipeline SVM Processor and the Scalable Semi-Parallel KNN Processor are the two hardware processors proposed for stress monitoring systems based on physiological data.

The scalable and pipelined architecture of the SVM processor uses four extracted features from heart rate and accelerometer sensors as input. The required coefficients were calculated offline and pre-computed weighted support vectors from trained model are loaded into memory blocks. The classifier performs dot product operations between the testing data and all supporting vectors to give a stress level indication as a result.

The KNN processor uses training samples and corresponding labels stored in a ROM block and finds the Euclidean distance between the testing sample and training data in parallel. The sorting block is responsible for finding the K smallest distance between the testing sample with all training data. The voting module generates the label of the testing sample based on the majority voting at the final step. The Finite State Machine (FSM) module is responsible for syncing and controlling all components in the design.

The parallelism of both processors allows them to be reconfigurable to process various number of features and different size of training data.

The researchers implemented the flexible on-board processors on ASIC and FPGA platforms. The performance was compared to existing embedded platforms, and it was found that the ASIC/FPGA implementations had the highest throughput (decision/sec) and lowest power consumption over all other platforms. The experimental results showed that the post-layout (ASIC) implementation of the SVM and KNN processors minimizes power consumption and latency as well as maintaining a low-area footprint for personalized stress monitoring. The ASIC implementation improves the energy efficiency by 42x and 12x over FPGA platform for SVM and KNN implementations, respectively.



Block Diagram of the proposed reconfigurable processors using SVM (a) and KNN (b) for personalized stress detection system.

2 Accelerator Implementation

2.1 Block Diagram

Figure 1 presents the main block diagram of the accelerator, and the following figures (2-4) are the main subunits that form the main diagram in figure 1, as it will be detailed in 2.2.

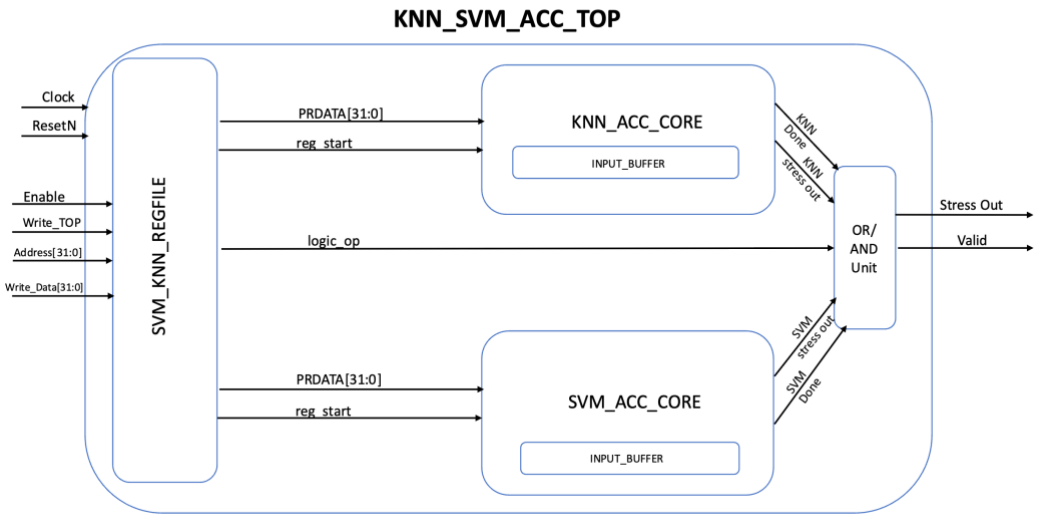


Figure 1 : Top Block Diagram

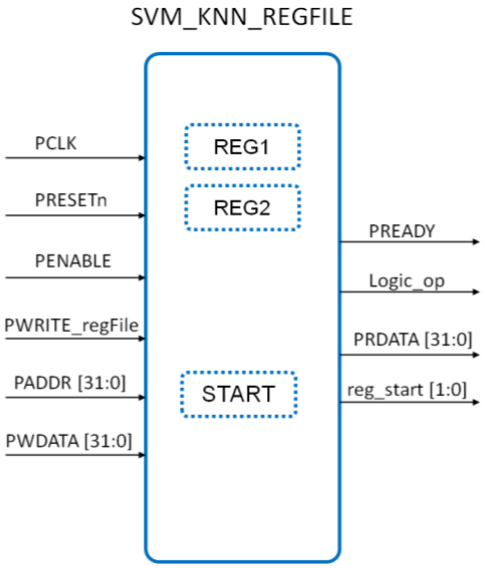


Figure 2 : SVM_KNN_REGFILE

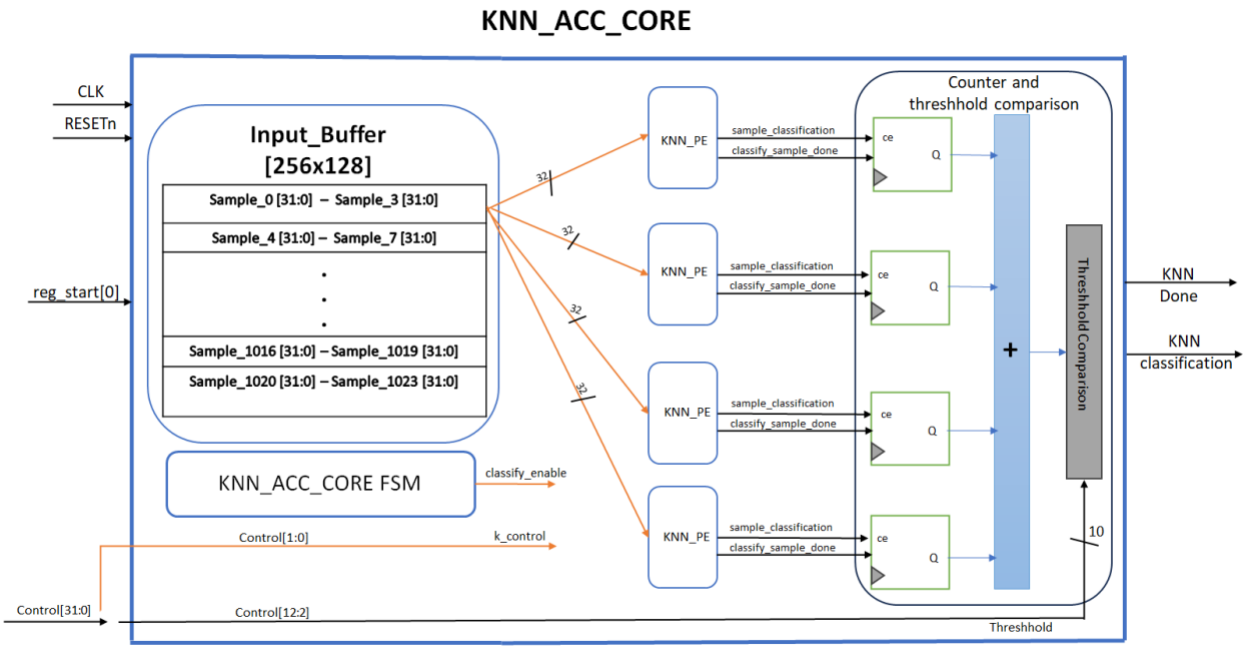


Figure 3 : KNN_ACC_CORE

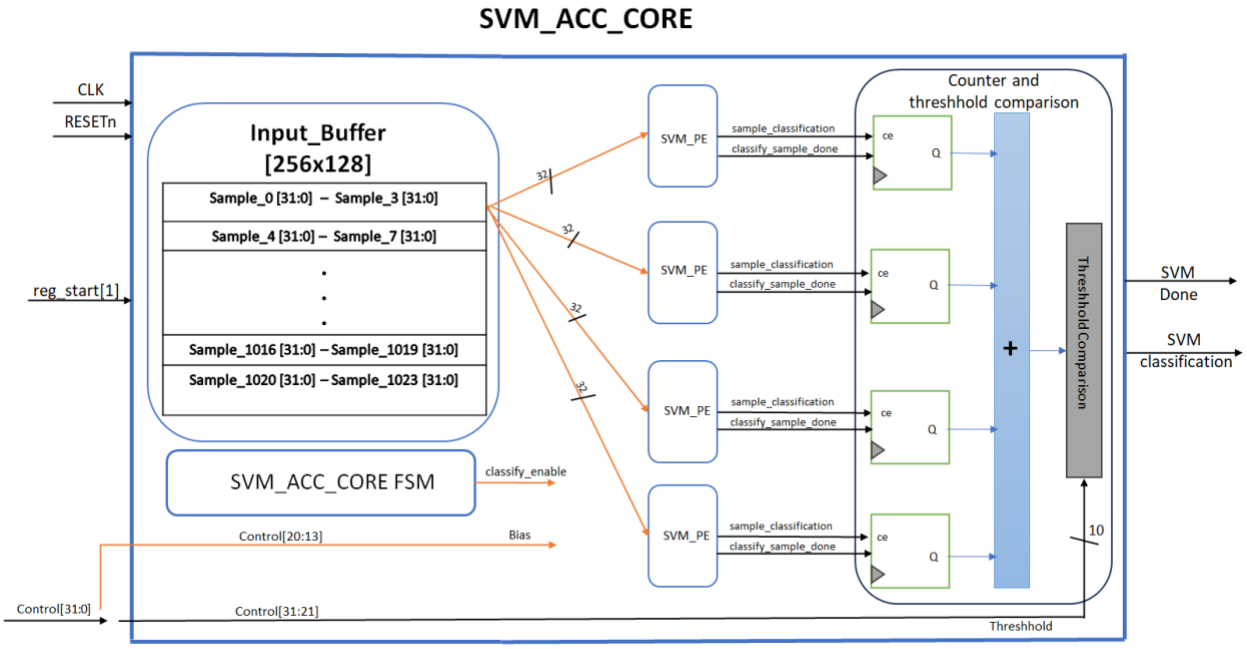


Figure 4 : SVM_ACC_CORE

2.2 Block Description

2.2.1 **Figure 1**, Top-level Block Diagram:

The top-level diagram encapsulates the foundational units of the accelerator, namely SVM_KNN_REGFILE, KNN_ACC_CORE, SVM_ACC_CORE, and the OR/AND unit. The operational flow begins with the SVM_KNN_REGFILE, serving as the central storage for loading essential parameters. Inputs such as Enable, Write_TOP, Address, and Write_Data facilitate the initialization of the system (further details about the interface is specified in 2.3 & 2.5).

Upon parameter loading, the SVM_KNN_REGFILE provides relevant information to each unit:

1. reg_start (2-bit): An output signal serving as a flag to determine the participation of cores in the classification process and to trigger the initiation of the operation.
2. PRDATA (32-bit): An output signal comprising control parameters for each core, which will be elaborated upon in subsequent sections.
3. logic_op (1-bit): An output signal specifying the logic operation to be executed by the OR/AND unit on the classification results of the two cores.

Afterwards, the classification process begins, driven by the reg_start signal. The cores operate independently, concluding in two key output signals:

1. Stress_Out : Calculated by applying the chosen logic operation to the sub-results of the two cores.
2. Valid : An indicator signaling the readiness of the final result.

This modular approach to classification ensures effective coordination and communication among the accelerator's components.

2.2.2 **Figure 2**, SVM_KNN_REGFILE:

The SVM_KNN_REGFILE serves as an essential component in the accelerator, acting as a central storage for necessary parameters crucial to its proper functionality. Comprising two registers—an initial 32-bit register, reg1, and a single-bit register, reg2—it facilitates parameter loading and customization as will be explained in section 2.9.1.

The regfile interface is managed by the APB protocol (refer to Section 1.3). The following signals define its interface:

Inputs:

1. PCLK, PRESETn: Clock and reset signals.
2. PENABLE: Enables reading or writing to the regfile.
3. PWRITE_regfile: Indicates the operation—writing to or reading from the regfile.
4. PADDR [31:0]: Specifies the address for reading/writing.
5. PWDATA [31:0]: Data to be written.

Outputs:

1. PREADY: Signals the readiness of the reading/writing result.
2. logic_op: Output signal specifying the logic operation for the OR/AND unit (0 for OR, 1 for AND).
3. PRDATA [31:0]: Output signal containing control parameters for each core:
   - [1:0] k_control: Determines the 'K' value for the KNN algorithm.
   - [12:2] knn threshold: Classification threshold.
   - [20:13] bias: Value of the bias for the SVM algorithm.
   - [31:21] svm threshold: Classification threshold.
4. reg_start [1:0]: A flag determining core participation in the classification process and triggering operation initiation. The LSB is for the KNN core and the MSB is for SVM core.

This modular regfile design offers user flexibility in configuring accelerator functionality, allowing parameter customization for tailored operation.

2.2.3 **Figure 3 & 4,** KNN_ACC_CORE and SVM_ACC_CORE:

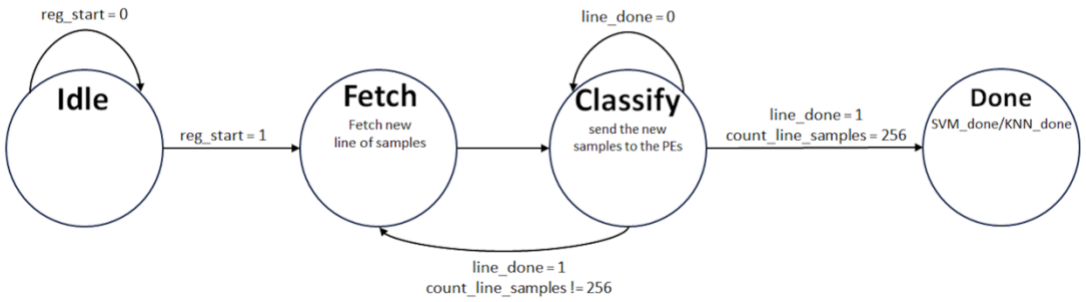
The SVM and KNN cores share a common structural framework, emphasizing parallel processing and user-configurable thresholds for optimal classification performance. Each core incorporates an Input Buffer, utilizing a RAM structure of 256x128 dimensions. With each row of the RAM housing four samples, each 32 bits in length, this design choice ensures the independence of input data, allowing parallel operation and preventing interference or dependencies between cores.

Central to the functionality of each core are the four Processing Elements (PEs), denoted as KNN_PEs/SVM_PEs. These PEs enable parallel processing by individually handling 25% of the input data. Maintaining dedicated copies of essential preliminary information, each PE operates autonomously, contributing to accelerated and efficient classification tasks without reliance on shared resources.

In the culmination of individual Processing Elements (PEs), the Counter and Threshold Comparison Unit is responsible for tallying the count of samples classified as '1' (indicating stress) and subsequently comparing this count to a user-defined threshold. This user-defined threshold mechanism, offering a customizable parameter, gives users the option to influence the decision-making process. Notably, for the KNN_CORE, the threshold is conveyed in the [12:2] bit range of the input control signal, while the SVM_CORE utilizes the [31:21] bit range. This distinctive feature allows users to finely adjust stress classification outcomes by specifying the minimum number of samples required to label the final result as '1' (stressed). Such mechanism enhances the versatility of both cores, making them adaptive to various application scenarios and responsive to individual user preferences.

Operating under the control of dedicated state machines, each core manages the fetching of new samples for the PEs and triggers the initiation of their classification tasks. This design ensures independence and synchronization, contributing to a cohesive and efficient operation of the SVM and KNN cores within the accelerator architecture.

SVM_ACC_CORE FSM / KNN_ACC_CORE FSM



Signal	Description
reg_start = 0	Loading the needed data to the RAMs.
reg_start = 1	All the needed data is loaded, and the cores are ready to work.
line_done = 0	PEs still working.
line_done = 1	All the PEs are done classifying the current sample.
line_done = 1 count_line_samples = 256	Done with all the 1024 sample (256 lines)
line_done = 1 count_line_samples != 256	Done with the current line of samples (classified) and need to move to the next line

Interface of SVM core:

Inputs:

- CLK, RESETn: Clock and reset signals.
- reg_start: single bit signal that determines whether the SVM core should take part in the classification process or not.
- Control [31:0] : provided from the output signal PRDATA of the regfile. Relevant bit range for the SVM core : [20:13] - value of the bias for the SVM algorithm,
  [31:21] - classification threshold.

Outputs:

- SVM_classification: final classification result of the SVM core.
- SVM_done: indicates that the SVM core finished its classification process and the result is ready.

Interface of KNN core:

Inputs:

- CLK, RESETn: Clock and reset signals.
- reg_start: single bit signal that determines whether the KNN core should take part in the classification process or not.
- Control [31:0] : provided from the output signal PRDATA of the regfile. Relevant bit range for the KNN core : [1:0] - value of the ‘K’ for the KNN algorithm,
  [12:2] - classification threshold.

Outputs:

- KNN_classification: final classification result of the KNN core.
- KNN_done: indicates that the KNN core finished its classification process and the result is ready.

2.3 Pins Description

Name	Input/Output	Width (bits)	Description
Clock	Input	1	Clock signal
ResetN	Input	1	Reset Signal
Enable	Input	1	Enables reading/writing to REGFILE
Write_TOP	Input	1	1-write to REGFILE, 0-read from REGFILE
Address	Input	32	Address of reading/writing
Write_data	Input	32	Data for writing
Stress_out	Output	1	Final result
Valid	Output	1	Indicates that the final result is ready

2.4 Clocks And Resets

The accelerator relies on well-coordinated clock signals and a carefully designed reset mechanism. This section examines the crucial elements that control the system's timing.

2.4.1 Clock Signal

At the core of the module, a singular clock signal pulsates with a frequency of 200 MHz, corresponding to a period of 5 ns. This synchronization strategy ensures that each sub-unit embedded within the accelerator operates in unison, adhering rigorously to the rhythmic beat of the shared clock. The synchronous execution of tasks across all sub-units guarantees a harmonized and efficient functioning of the entire system, optimizing performance and minimizing latencies.

2.4.2 Reset Mechanism

In parallel to the clock signal, the accelerator is equipped with a robust reset mechanism vital for system stability and flexibility. The reset signal operates asynchronously, triggered by the negative edge of the clock signal. This design gives the accelerator with the capability to reset its internal state swiftly and effectively, offering a responsive means to recalibrate the system whenever necessary. The asynchronous nature of the reset signal ensures that it can be asserted independent of the clock cycle, contributing to the adaptability and resilience of the accelerator in dynamic operational scenarios.

2.5 Interface Description

The accelerator incorporates four interfaces to facilitate seamless interaction with external user, ensuring compatibility with industry standards:

1. Clock/ResetN Interface : Comprising two fundamental signals, namely the clock and resetN, this interface is responsible for the temporal synchronization and resetting of the accelerator. Refer to Section 2.4 for an in-depth exploration of these signals.

2. APB Interface : Includes signals tailored for both writing to and reading from the REGFILE, the APB Interface establishes a standardized communication protocol between external components and the accelerator. Further elucidation of this interface is available in Section 1.2.

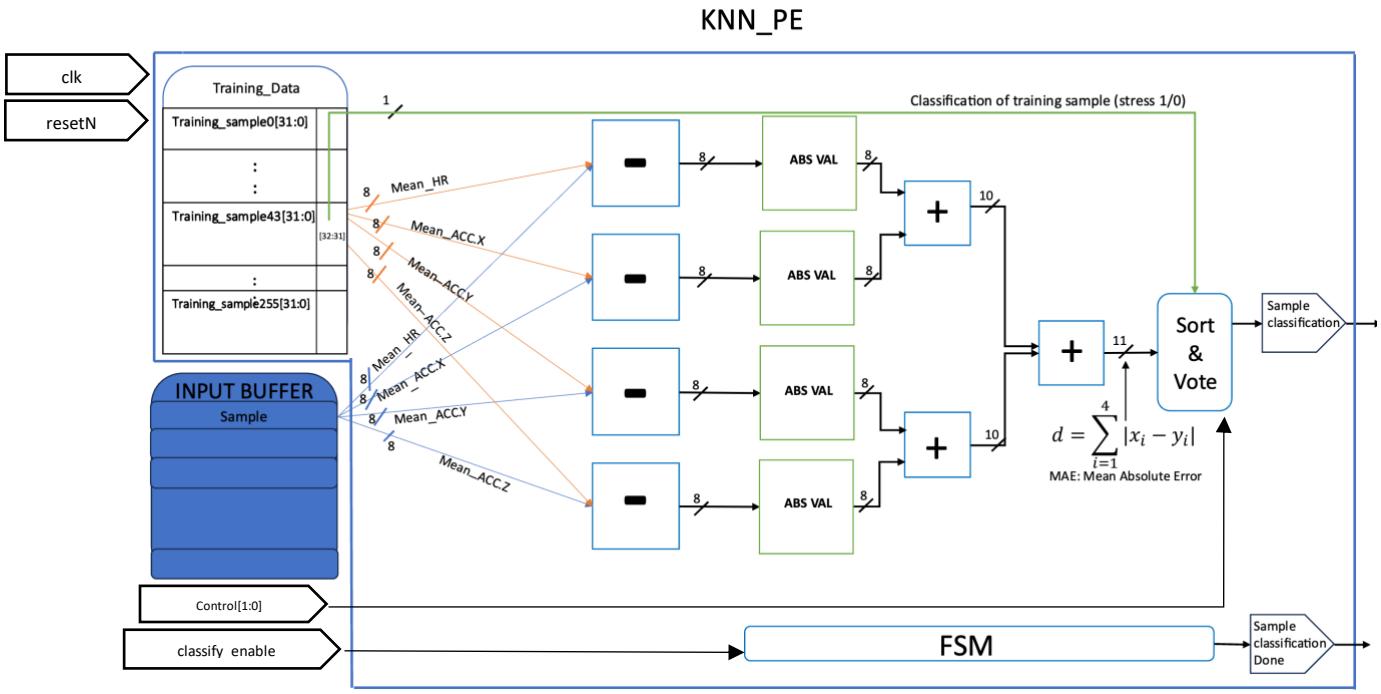
3.1. Stress_out Signal : Functioning as an output signal, Stress_out encapsulates the conclusive outcome of the accelerator's computation, differentiating between a state of stress (assigned '1') and a non-stressed state (assigned '0').

3.2. Valid Signal : Functioning as an output signal, Valid indicates that the final result in the Stress_out signal is ready to be read.

4. DMA Stub Interface : This interface acts as a bridge for efficient data transfer between the accelerator and external memory using Direct Memory Access (DMA). It includes dedicated signals for commands, memory addresses, and data flow direction.

2.6 Sub-units Description

2.6.1 KNN Processing Element

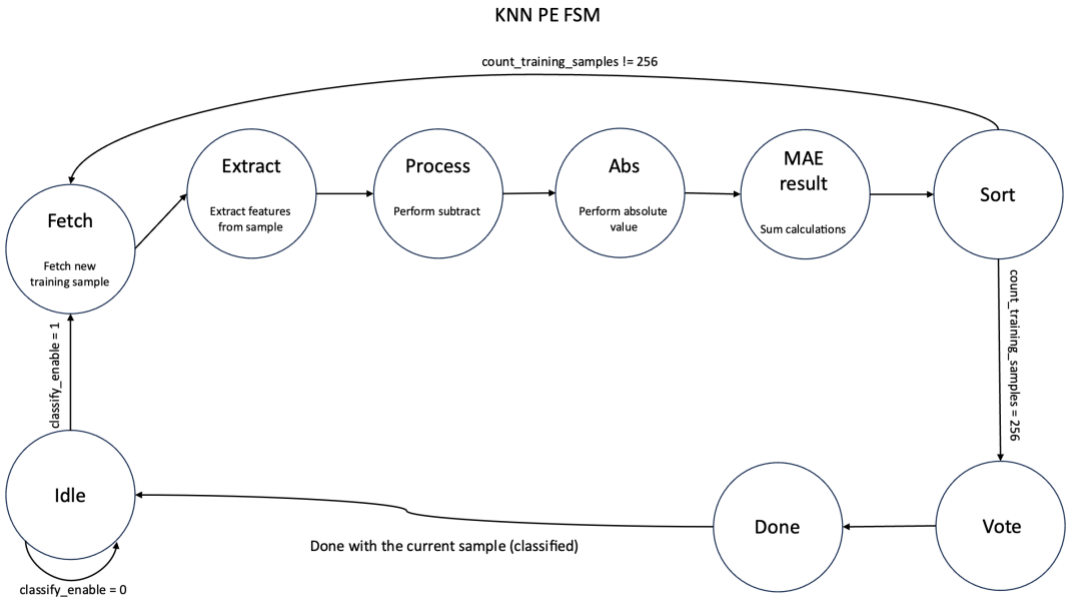


The K Nearest Neighbors (KNN) Processing Element (PE) is a fundamental component within the proposed hardware accelerator for personalized stress detection, comprising four individual PEs, each tasked with classifying 25% of the data in the input buffer.

The architecture of each PE centers around a singular 256X33 Random Access Memory (RAM) module named Training Data, housing essential training samples. Each sample, divided into four features, is represented by 8 bits. The PE's feature processing involves transferring each feature to an execution unit for Mean Absolute Error (MAE) distance calculation, optimizing computational efficiency over traditional Euclidean distances.

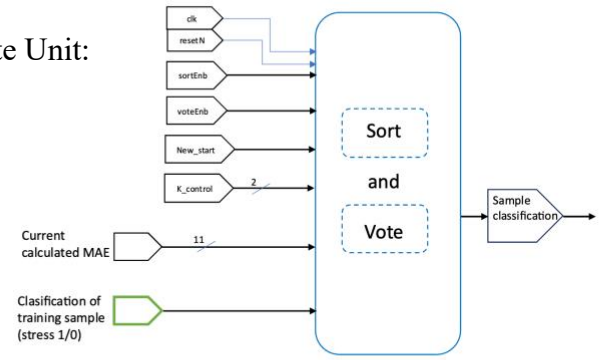
The computed MAE distances are summed, resulting in a cumulative distance value that is subsequently utilized by the Sort&Vote mechanism to determine the K nearest neighbors. Notably, the 33rd bit, representing the classification type of the current sample in the training data, is used exclusively during the voting phase, contributing to the decision on the majority class.

The KNN PE is controlled by the following state machine:

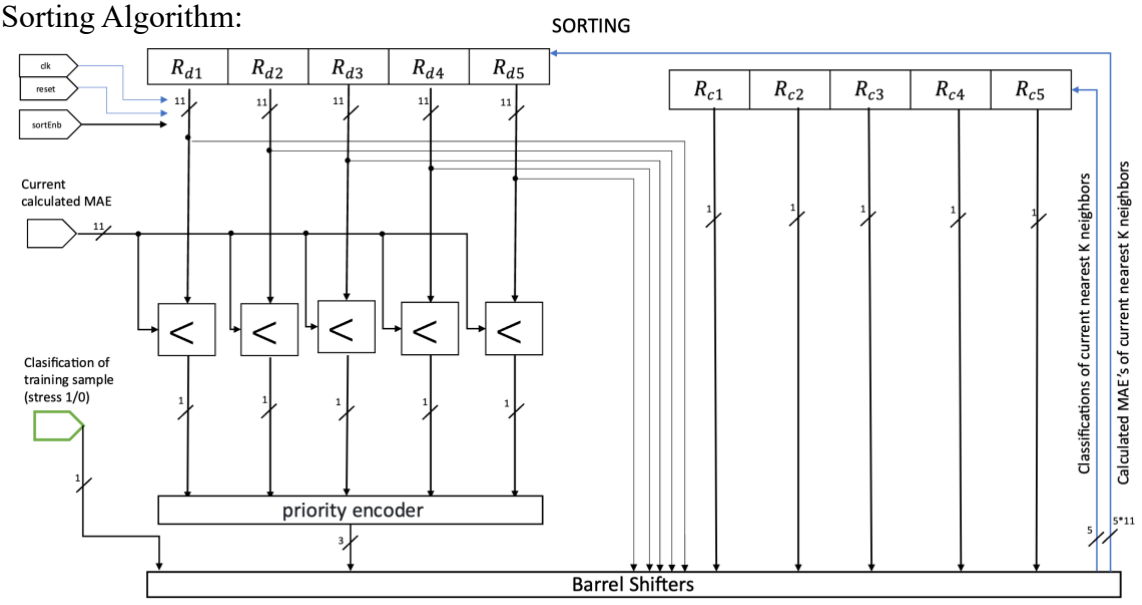


The PE remains in an idle state, awaiting activation signaled by the classify_enable signal transitioning to '1'. Upon activation, the FSM enters the 'Fetch' state, initiating the retrieval of a new training sample to start a new calculation. Subsequently, the 'Extract' phase isolates four 8-bit features from the fetched line, followed by individual subtract calculations for MAE distance in the 'Process' state. The 'ABS' state performs the absolute value computation, and the resulting distances from the four features are aggregated in the 'MAE_result' state. Upon MAE result completion, the 'Sort' state executes the sorting phase, identifying the k nearest neighbors while concurrently tracking the count of fetched training samples. If the count_training_samples remains below 256, indicating incomplete traversal of all training samples, the FSM returns to the 'Fetch' state. Conversely, if the count_training_samples reaches 256, signifying completion of k nearest neighbors determination for the current sample, the FSM transitions to the 'Vote' state, contributing to the majority class decision. Upon completion of the classification for the current input buffer sample, the system returns to the 'idle' state, ready for subsequent sample classification.

Sort and Vote Unit:

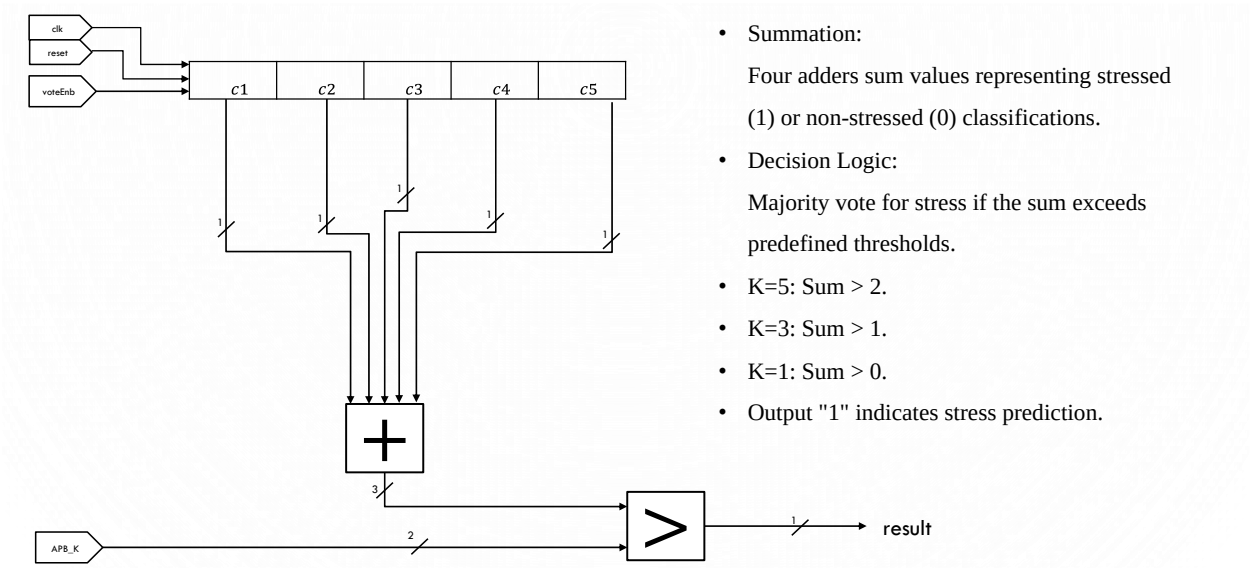


Sorting Algorithm:



The sorting algorithm implemented in the PE efficiently determines the K nearest neighbors. To optimize both space and time complexity, the system utilizes five registers, Rd1-Rd5, initialized to represent the largest possible distance. These registers store the five closest distances to the current sample in ascending order. The priority encoder, upon comparing each new distance with those stored in the registers, determines whether the new calculation enters the list of closest distances. The barrel shifters then adjust the registers to maintain the five closest distances in ascending order. Additionally, five registers, Rc1-Rc5, initialized to zero, store the classification type of samples with the closest distances. The barrel shifters synchronize shifts for both distance and classification registers. After processing all training samples, the system retains the five closest distances with their respective classifications, transitioning to the subsequent voting stage.

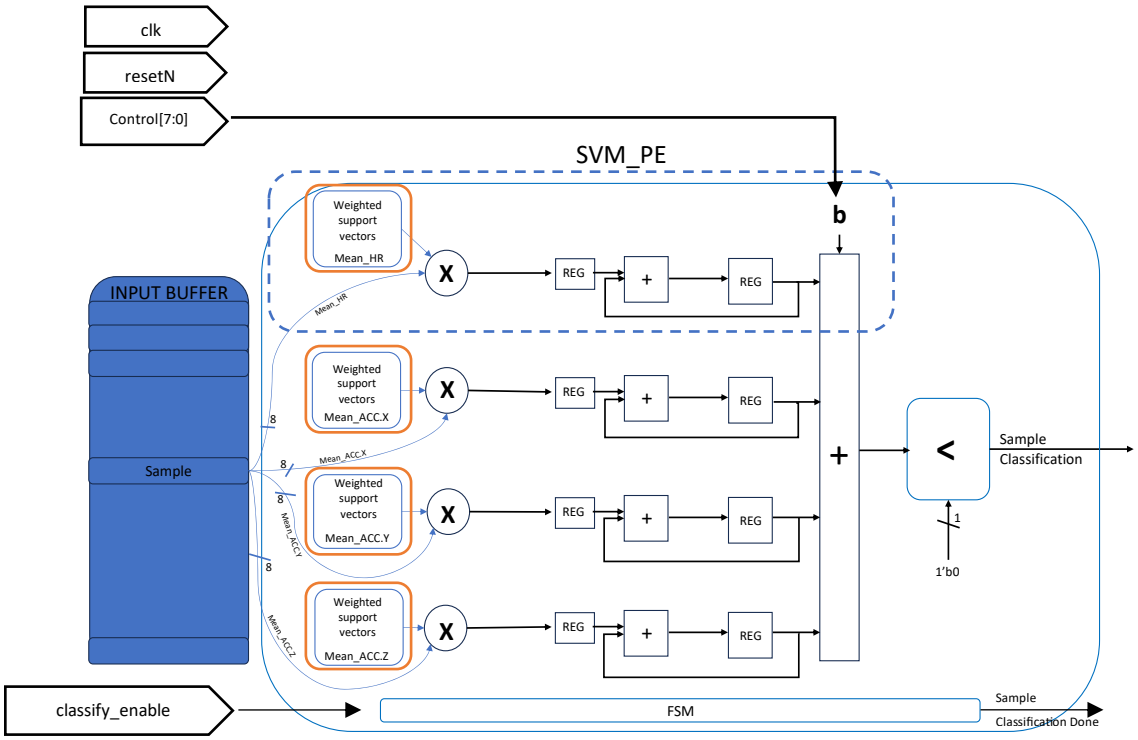
Voting Stage:



- Summation:
  Four adders sum values representing stressed (1) or non-stressed (0) classifications.
- Decision Logic:
  Majority vote for stress if the sum exceeds predefined thresholds.
- K=5: Sum > 2.
- K=3: Sum > 1.
- K=1: Sum > 0.
- Output "1" indicates stress prediction.

During the voting stage, the system employs an adder to sum the five values representing stressed (1) or non-stressed (0) classifications. For a K value of 5, a sum greater than 2 indicates a majority vote for stress, signifying that at least three of the five closest samples are classified as stressed. For K values of 3 or 1, the decision is based on whether the sum is greater than 1 or 0, respectively. An affirmative result indicates a majority vote for stress, leading to the output "1" and an increment in the relevant counter, reflecting the prediction of stress for the current sample.

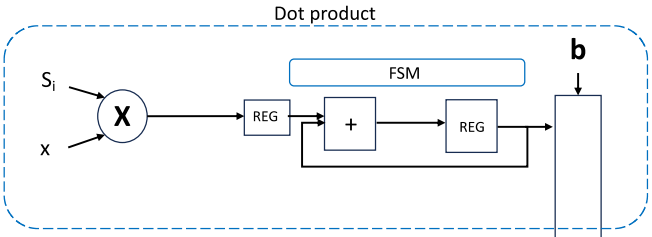
2.6.2 SVM Processing Element

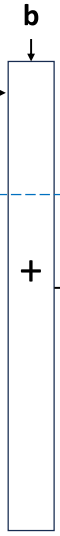


Within the SVM core, the SVM Processing Element (SVM PE) plays a crucial role as one of the four processing elements, each individually responsible for 25% of the data within the input buffer.

Before delving into the details of the classification process, it is essential to explain how the support vectors are stored and managed. The SVM PE employs four storage units highlighted in orange, collectively forming a single RAM with dimensions 64x32. Each support vector is constructed from four 8-bit features, amounting to a total of 64 support vectors.

Notably, these vectors are stored post-multiplication by their corresponding weights. This storage approach is permissible due to the nature of the operations conducted on the input and weighted support vectors—predominantly multiplications and sums. The rearrangement of these operations does not alter the calculation results, enabling an efficient handling of support vectors within the SVM core.



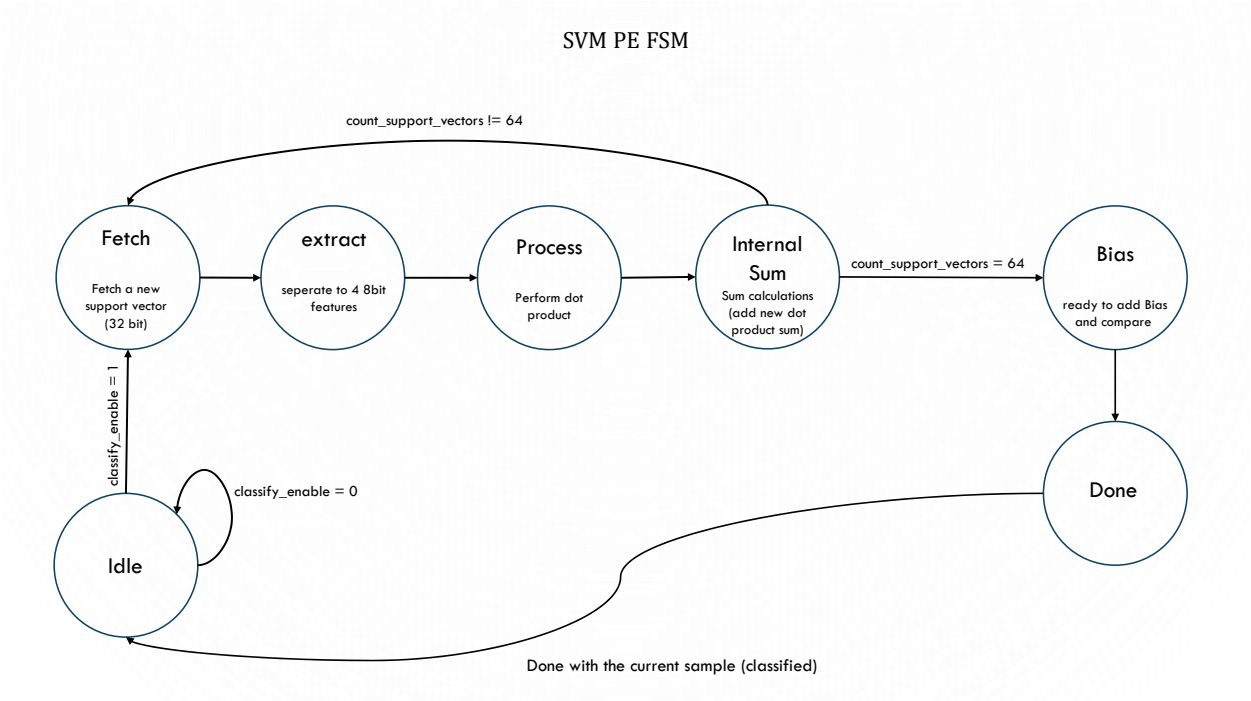


$$f(x) = sign(\sum_{i=1}^{num_s} (S_i \cdot x) + b)$$

- S_i is the value that represents the feature of the support vector 'i' .
- x is the feature of the new sample.
- x and S_i are the of same category of features.
- We have 4 dot products; each handles a different feature.

- Sum is done on all dot products between the features of the new sample and the features of **ALL** stored weighted support vectors.
- Bias is added before the classification decision which is determined by the sign of the result.

Controlling the flow of operations inside the SVM PE is the following state machine :



The FSM governing the SVM Processing Element (SVM PE) operates in distinct states. In the 'Idle' state, it awaits activation signaled by the classify_enable signal. Upon activation, it transitions to the 'Fetch' state, retrieving a new support vector for calculation. The 'Extract' state

isolates four 8-bit features from the fetched data, followed by the 'Process' state, which performs the SVM algorithm's dot product. In the 'Internal Sum' state, the result is added to the cumulative sum. If 'count_support_vectors' is less than 64 , signifying incomplete processing, the FSM returns to 'Fetch'; otherwise, it moves to 'Bias,' adding the bias parameter. The result is then compared to zero; if greater, the sample is 'stressed'; if smaller, 'not stressed.' The FSM enters 'Done,' signaling completion, before returning to 'Idle' for the next classification cycle signaled by classify_enable.

2.7 Performance

In this section, the theoretical exploration of the VLSI accelerator's performance is unveiled through a detailed pipeline diagram. Illustrating sequential stages, the diagram provides insights into the operational efficiency made to optimize speed and efficacy. The focus lies on presenting a concise overview that establishes the groundwork for subsequent empirical evaluations and real-world applications. It is essential to note that measurements in this analysis were conducted under the worst-case scenario, where both cores were concurrently operational. The test protocol ensured that both cores systematically processed all input buffer samples (1024) before delivering the classification result.

Note: each cycle is set to be 5ns.

2.7.1 Pipeline Diagram

KNN_CORE – while the classification (reg_start on) start on cycle i.

cycle	Fetch Fetch new line of samples (4 samples to classify)	Classify Fetch, Extract, Process, ABS, MAE_res, Sort, Vote 6(states)x256(training samples) = 1536	ce, Q PEs_Done	+	Threshhold Comparison
i	[Sample_1]				
i+1→i+1537		[Sample_1]			
i+1538	[Sample_2]		[Sample_1]		
i+1539		[Sample_2]		[Sample_1]	
i+1540		[Sample_2]			[Sample_1]
i+1541→i+3075		[Sample_2]			
i+3076	[Sample_3]		[Sample_2]		
i+3077		[Sample_3]		[Sample_2]	
i+3078		[Sample_3]			[Sample_2]

Performance Calculation Inside the KNN PE ('classify' state in the KNN core):

- The FSM inside the KNN PE comprises 9 different states, with 6 being constantly repeated in the classification phase of a new sample.
- For a new sample to be classified, it traverses all 256 training samples, each going through the 6 states mentioned above before eventually reaching the 'vote' state, and with every state taking one cycle, this results in 6*256 +1 = 1537 cycles.
- Considering every cycle is 5ns, processing time of a single sample is: 5*1537 = 7685ns.
- Total processing time for classifying all the samples in a single PE: 7685*256 = 1,967,360ns.

SVM_CORE – while the classification (reg_start on) start on cycle i.

cycle	Fetch Fetch new line of samples (4 samples to classify)	Classify Fetch / Extract / Process / Internal_Sum / Bias 4(states)x64(sv) = 256	ce / q PEs_ Done	+	Threshhold Comparison
i	[Sample_1]				
i+1→i+257		[Sample_1]			
i+258	[Sample_2]		[Sample_1]		
i+259		[Sample_2]		[Sample_1]	
i+260		[Sample_2]			[Sample_1]
i+261→i+515		[Sample_2]			
i+516	[Sample_3]		[Sample_2]		
i+517		[Sample_3]		[Sample_2]	
i+518		[Sample_3]			[Sample_2]

Performance Calculation Inside the SVM PE ('classify' state in the SVM core):

- The FSM inside the SVM PE comprises 7 different states, with 4 being constantly repeated in the classification phase of a new sample.
- For a new sample to be classified, it traverses all 64 support vectors, each going through the 4 states mentioned above before eventually reaching the 'bias' state, and with every state taking one cycle, this results in
  4*64+1 = 257 cycles.
- Considering every cycle is 5ns, processing time of a single sample is: 5*257 = 1285ns.
- Total processing time for classifying all the samples: 1285*256 = 328,960ns.

2.7.2 Latency

Latency = Total Processing Time / Number of Processed Samples.

Note: The total processing time calculated in the previous section was for a single PE, and since the 4 PEs of each core work in parallel, it is the same time for classifying all of the 1024 input samples.

KNN latency = $\frac{1{,}967{,}360[ns]}{1024[samples]} = 1921ns$

SVM latency = $\frac{328{,}960[ns]}{1024[samples]} = 321ns$

2.7.3 Throughput

Throughput = Number of Processed Samples / Processing Time.

Note: since the 4 PEs of each core work in parallel, four samples are classified per processing time.

KNN TP = $\frac{4[samples]}{7685[ns]} = \frac{4 \cdot 4[bytes]}{7.685 \cdot 10^{-6}[sec]} = 572{,}542\left[\frac{bytes}{sec}\right]$

SVM TP = $\frac{4[samples]}{1285[ns]} = \frac{4 \cdot 4[bytes]}{1.285 \cdot 10^{-6}[sec]} = 3{,}424{,}124[\frac{bytes}{sec}]$

2.8 Synthesis

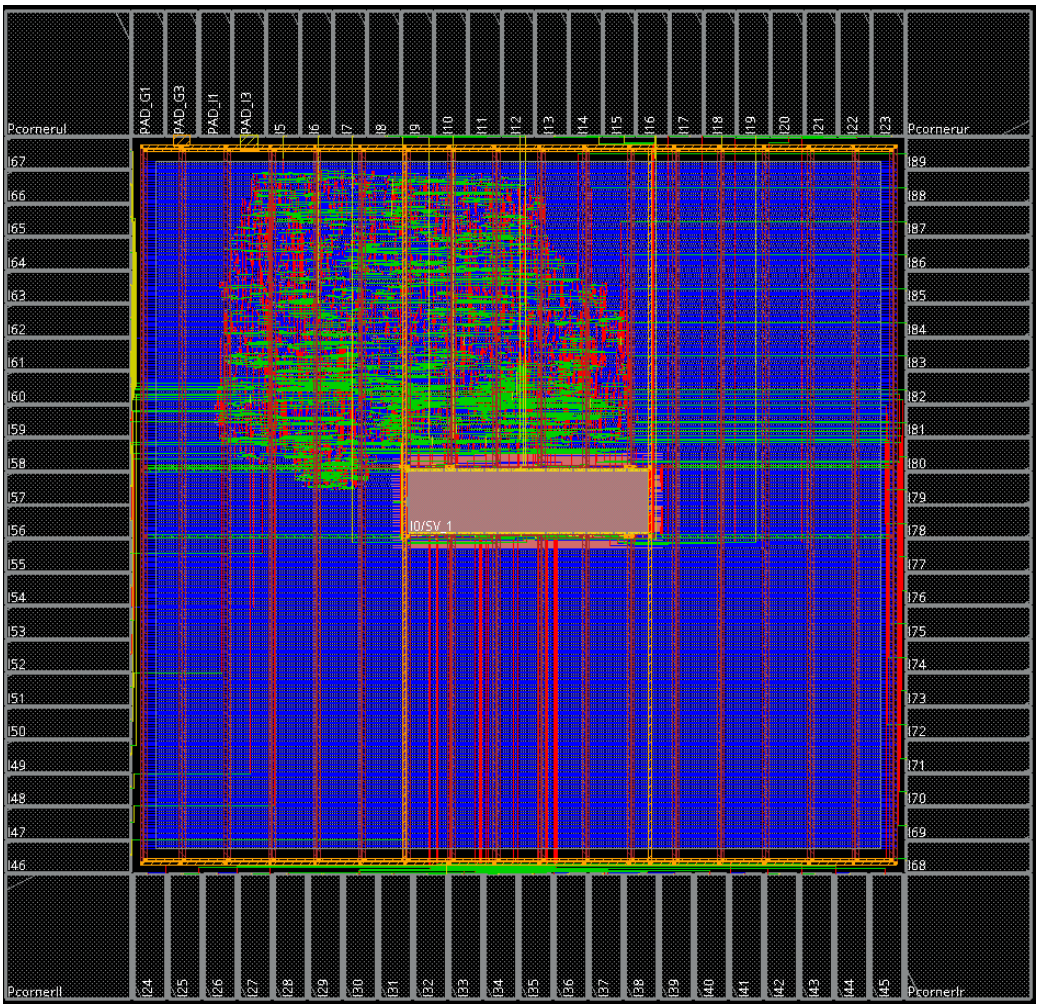
Synthesis and layout were done on a single SVM PE unit.

2.8.1 Technology And Constrains

The synthesis and layout processes were performed using the tools Synopsys and Cadence (Design Vision and Innovus). The layout result was constructed using the Innovus software (Cadence) with the Tower 0.18u design kit.

2.8.2 Floorplan

Layout result : the chip and its dimensions 2048um:1985um.



Floorplan description:

The floorplan of the layout provides a visual representation of the chip's spatial organization, revealing crucial insights into the design's structure and utilization.

The distinct colors across the chip signify different metal layers that collectively form the integrated circuit.

The predominant usage of green and red shades highlights the extensive deployment of combinatorial logic, emphasizing the substantial computational elements.

The central area is dedicated to the memory unit, symbolized by its distinctive color, indicating a focal point for storage and retrieval operations.

However, the overall chip size is predominantly influenced by the number of input and output connections (I/O), making it pad-limited. Interestingly, despite the importance of combinatorial logic, it occupies a relatively small portion of the chip compared to the space reserved for I/O. This configuration suggests that the primary bottleneck in chip size arises from the need to efficiently manage external connectivity.

2.8.3 Area And Power

In the synthesis and layout phase for a single SVM PE, the obtained area and power results offer a comprehensive view of the design's composition and efficiency.

Area results:

```
Number of ports:                        1011
Number of nets:                         4254
Number of cells:                        3105
Number of combinational cells:          2701
Number of sequential cells:              379
Number of macros/black boxes:              1
Number of buf/inv:                       541
Number of references:                     38

Combinational area:               3969.250000
Buf/Inv area:                      352.500000
Noncombinational area:            2574.250000
Macro/Black Box area:             5032.600098
Net Interconnect area:            1613.647999

Total cell area:                 11576.100098
Total area:                      13189.748097
```

- The "Combinational area" (3969.25 square units) signifies the space dedicated to logic elements performing operations solely based on current inputs.
- The "Buf/Inv area" (352.5 square units) encompasses the area occupied by buffers and inverters, often implemented using basic logic gates like NAND gates.
- The "Noncombinational area" (2574.25 square units) incorporates additional elements beyond pure combinational logic, including sequential logic, buffers, and inverters.
- The "Macro/Black Box area" (5032.6 square units) designates the space occupied by higher-level, pre-designed modules.
- The "Net Interconnect area" (1613.648 square units) represents the space allocated for wiring and interconnections facilitating communication between different components.

The total cell area is calculated at 11576.1 square units, resulting in a comprehensive total area of 13189.748097 square units for the synthesized SVM PE design

Power analysis:

```
Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  36.8651 mW   (99%)
  Net Switching Power  = 325.6653 uW    (1%)
                         ---------
Total Dynamic Power    =  37.1908 mW  (100%)

Cell Leakage Power     =   1.7257 uW
```

Power Group	Internal Power	Switching Power	Leakage Power	Total Power	(%)	Attrs
io_pad	0.0000	0.0000	0.0000	0.0000	(0.00%)	
memory	31.5944	5.4932e-03	1.5824e+06	31.6015	(84.97%)	
black_box	0.0000	0.0000	0.0000	0.0000	(0.00%)	
clock_network	0.0000	0.0000	0.0000	0.0000	(0.00%)	
register	5.1436	3.9497e-02	5.6803e+04	5.1832	(13.94%)	
sequential	0.0000	0.0000	0.0000	0.0000	(0.00%)	
combinational	0.1270	0.2807	8.6482e+04	0.4078	(1.10%)	
Total	36.8651 mW	0.3257 mW	1.7257e+06 pW	37.1925 mW		

2.8.4 Critical Path

```
clock CLK (rise edge)                                  5.00       5.00
clock network delay (ideal)                            0.00       5.00
dot_result_3_reg_15_/CP (decrq1)                       0.00       5.00 r
library setup time                                    -0.19       4.81
data required time                                                4.81
-------------------------------------------------------------------------
data required time                                                4.81
data arrival time                                                -4.81
-------------------------------------------------------------------------
slack (MET)                                                       0.00
```

The critical path analysis provides insights into the timing characteristics of the synthesized design.

The path encounters several logic gates, inversions, and multiplexers, each contributing to the overall delay. The slack, represented as "0.00", indicates that the design meets timing requirements, with a margin of zero between the data arrival time and the required time.

2.9 Programmer's Guide

This section provides a clear programming interface, configuration instructions, and practical usage examples, this guide empowers developers with the tools to seamlessly integrate the accelerator into diverse applications.

2.9.1 Registers Description

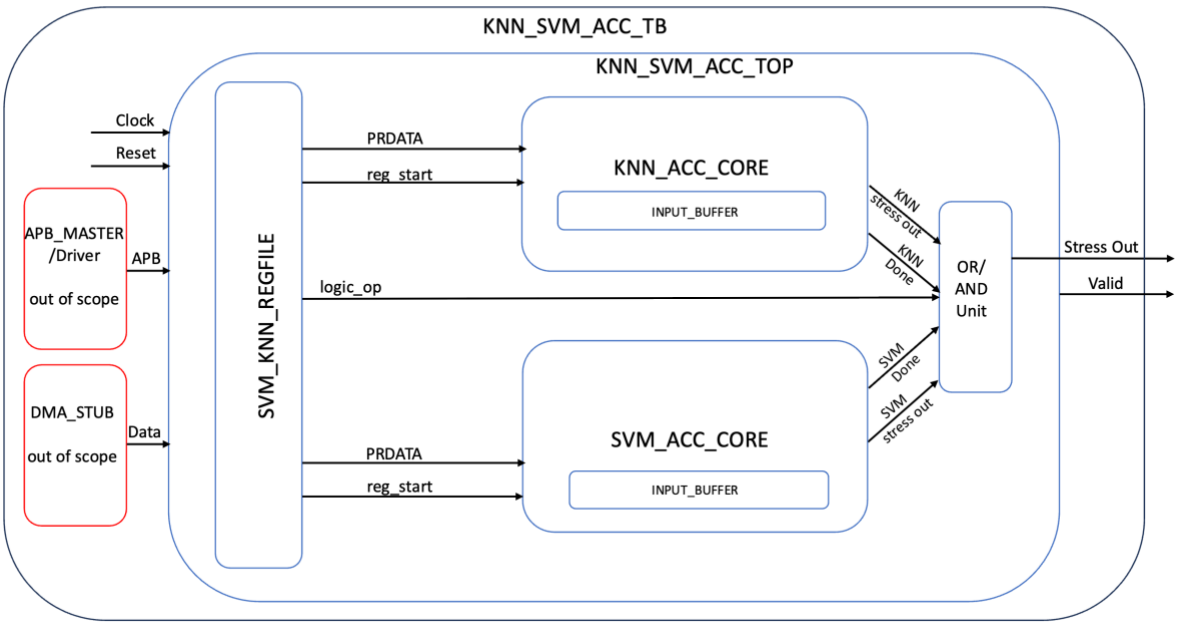
Register Name	Address	Description
reg1[31:0]	0x0970	Stores control parameters for the cores: • [1:0] k_control: Determines the 'K' value for the KNN algorithm. • [12:2] knn threshold: Classification threshold. • [20:13] bias: Value of the bias for the SVM algorithm. • [31:21] svm threshold: Classification threshold.
reg2	0x0974	Stores the chosen logic operation: • '0': OR • '1': AND
reg_start[1:0]	0x0978	Stores the core participation choice in the classification process: • '00': Both cores idle • '01': KNN core active, SVM core idle • '10': SVM core active, KNN core idle • '11': Both cores active

2.9.2 Usage Example

```
// ------------------------------------------------------------------------
//             Test Pattern
// ------------------------------------------------------------------------
initial begin

    // Load binary data into memories.

    // Initialize all components
    initiate_all;
    #100
    resetn = 1'b1;

    // Start processing - WRITE TO REG_START output 0 to initiate
    @(posedge clk or negedge resetn);
    write_to_reg(reg_start address, reg_start_data [31:0], enable = 1, write_enable = 1);
    #100

    // Set threshold values and control parameters for SVM and KNN by writing to reg 1
    @(posedge clk or negedge resetn);
    write_to_reg(reg1_address, reg1_data [31:0] , enable = 1, write_enable = 1);
    #100
    // Read threshold values and control parameters for SVM and KNN by reading from reg 1 and storing it in output read_data
    @(posedge clk or negedge resetn);
    read_reg(reg1_address,  enable = 1, write_enable = 0);
    #100

    //REG OPERATION OR-0 | AND-1
    // Set logical operation by writing to reg 2
    @(posedge clk or negedge resetn);
    write_to_reg(reg2_address, reg2_data [31:0] , enable = 1, write_enable = 1);
    #100
    // Read logical operation by reading from reg 2 and storing it in output reg_operation
    @(posedge clk or negedge resetn);
    read_reg(reg2_address,  enable = 1, write_enable = 0);
    #100

    //WRITE TO REG_START output: 01-KNN | 10-SVM | 11-both active
    @(posedge clk or negedge resetn);
    write_to_reg(reg_start address, reg_start_data [31:0], enable = 1, write_enable = 1);
end
```

3 Zero-order verification

3.1 Block Diagram:



The above diagram displays the testbench used to manage and test the functionality of the accelerator. In red the APB master and DMA stub can be seen connected to the accelerator as they provide control signals and data to the accelerator.

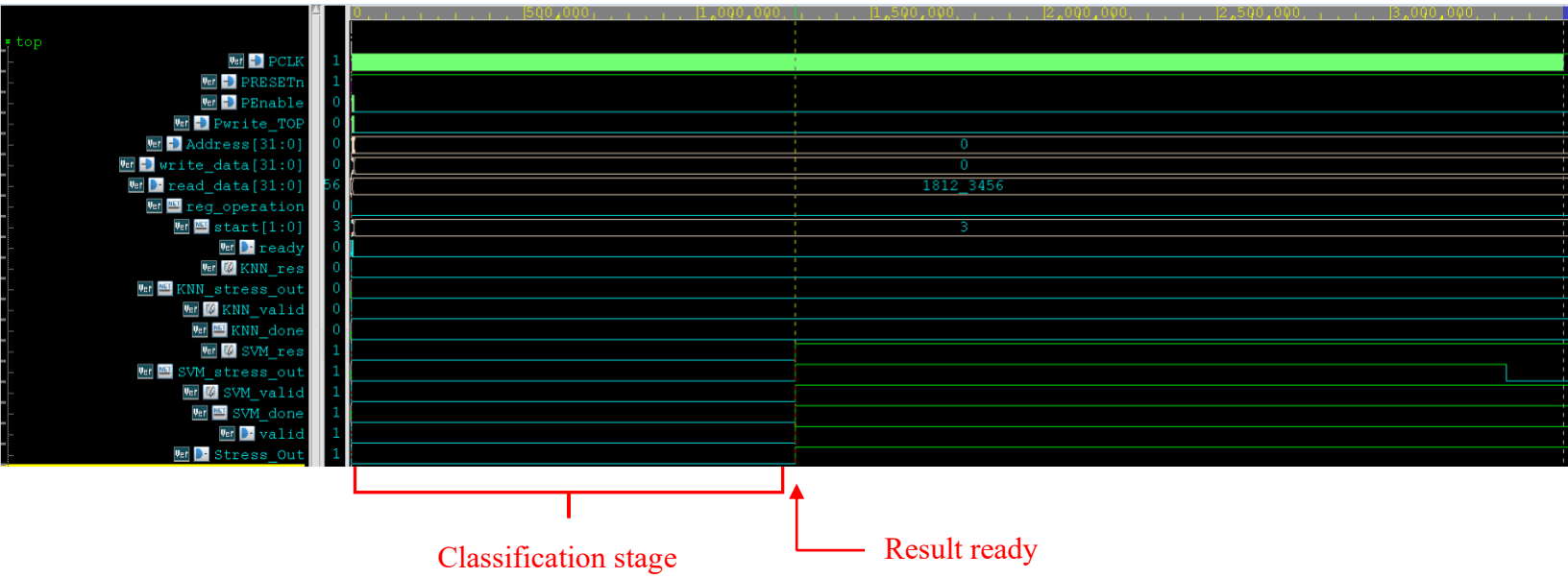
3.2 Test Plan:

During the testing stage, the primary objective was to verify the functionality and integrity of the system—a process commonly referred to as zero-order verification. These tests provided a comprehensive overview of every stage within the accelerator, explaining the details of extraction, calculation, and classification result retrieval. To facilitate this verification process, behavioral memories were employed and manually populated with randomly generated data tailored to fit the dimensions of each memory unit. This careful approach ensured that every aspect of the accelerator's operation was thoroughly inspected, laying the foundation for robust performance and reliable classification outcomes. Each testcase has its own pass criteria (threshold, logic operator etc.) specified in the next section.

3.3 Results:

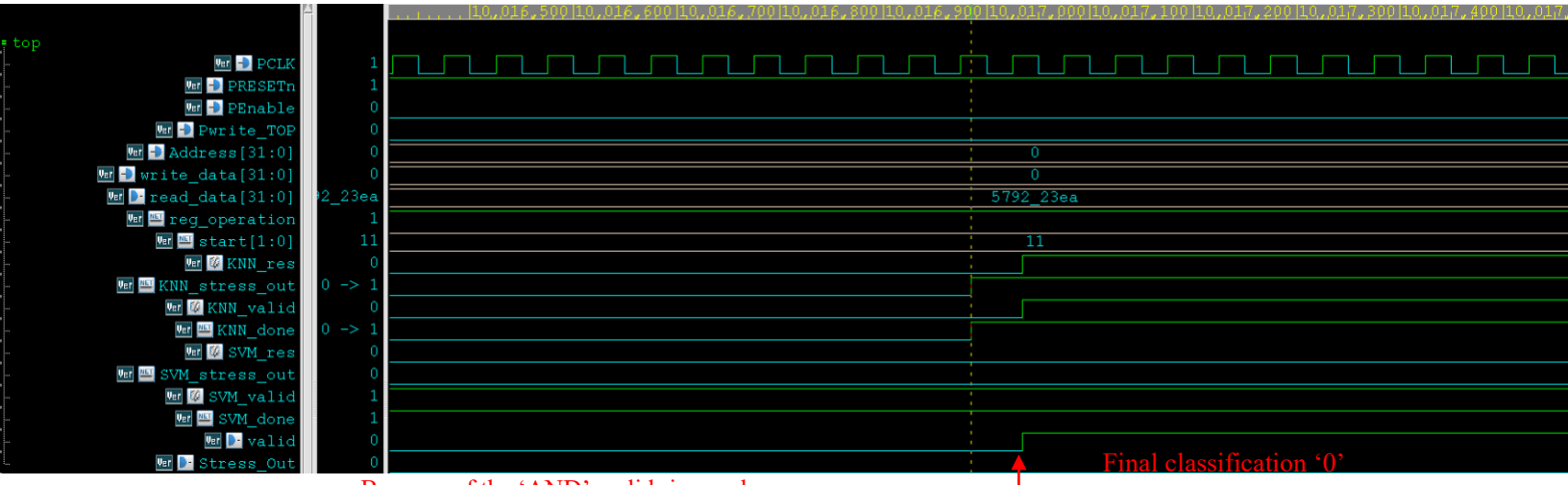
Top level:

Test case 1: SVM core is operational, therefore the final result is ready whenever the SVM classification result is ready. This result also represents the scenario where the SVM classification is '1' and the chosen logic operation is 'OR'.



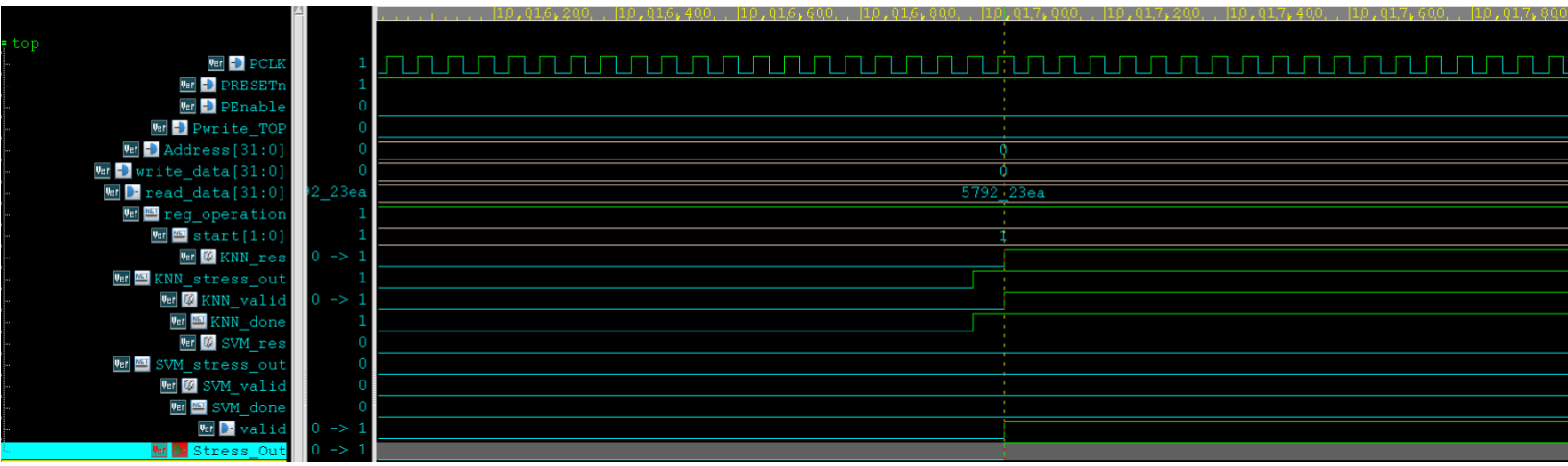
Test case 2:

- Both cores operational – reg_start = '11'.
- Threshold of the KNN core is 250, KNN classification is '1'.
- Threshold of the SVM core is 700, SVM classification is '0'.
- Logic AND between the result of the two cores → final result of classification is zero.



Test case 3:

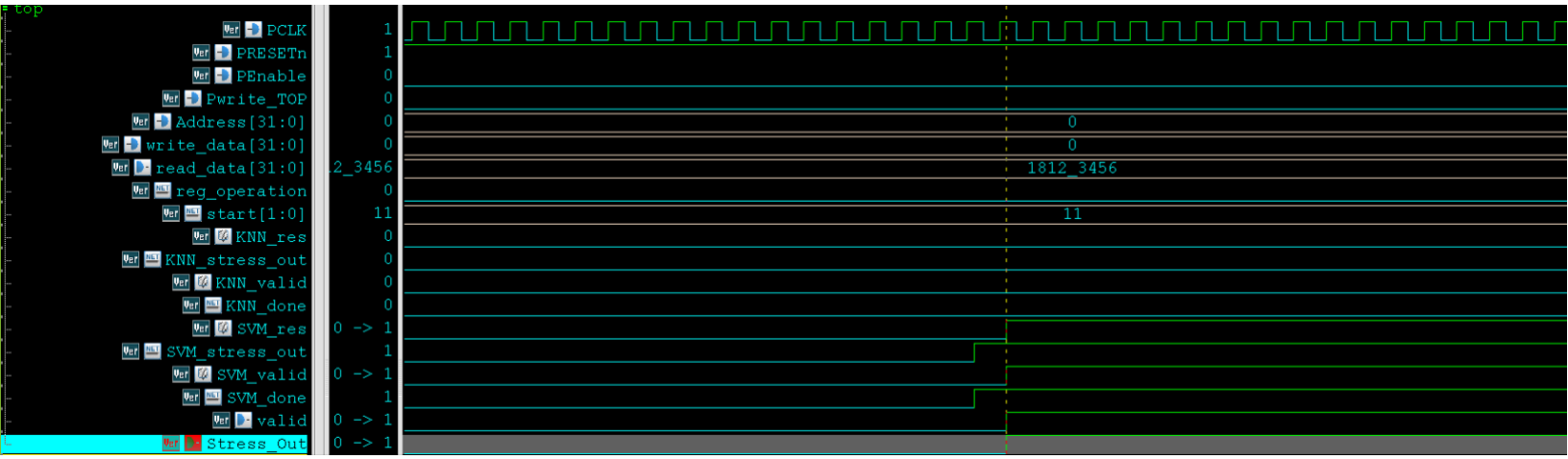
- KNN core operational – reg_start = ‘01’.
- Threshold of the KNN core is 250, KNN classification is ‘1’.
- Threshold of the SVM core is 700, SVM classification is ‘0’.
- Logic AND between the result of the two cores → final result of classification is one.



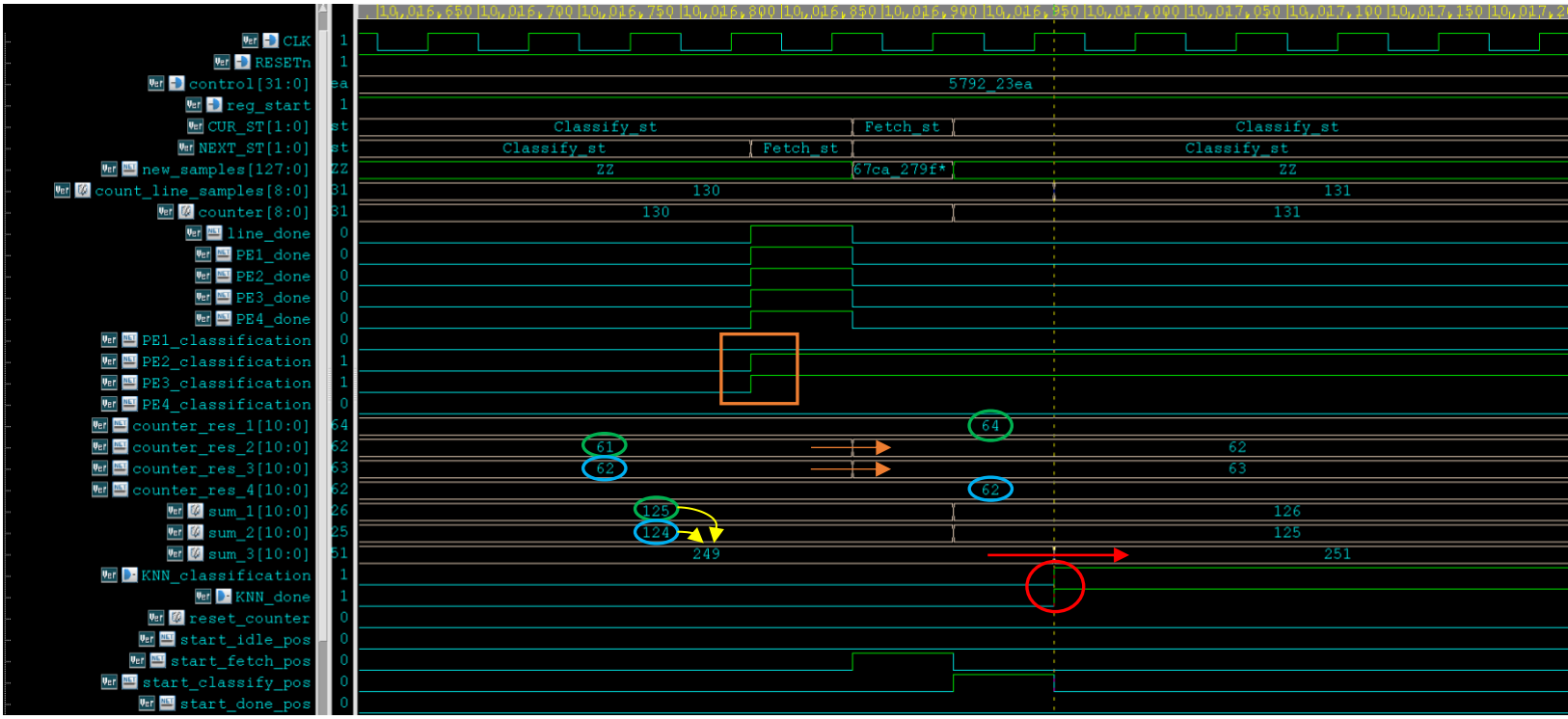
Despite the logic ‘AND’, and one of the results is ‘0’, the final result is ‘1’ because only the KNN core is considered operational.

Test case 4:

- Both cores operational – reg_start = ‘11’.
- Threshold of the KNN core is 1300, KNN classification is ‘0’.
- Threshold of the SVM core is 192, SVM classification is ‘1’.
- Logic OR between the result of the two cores → final result of classification is one.



KNN Core:



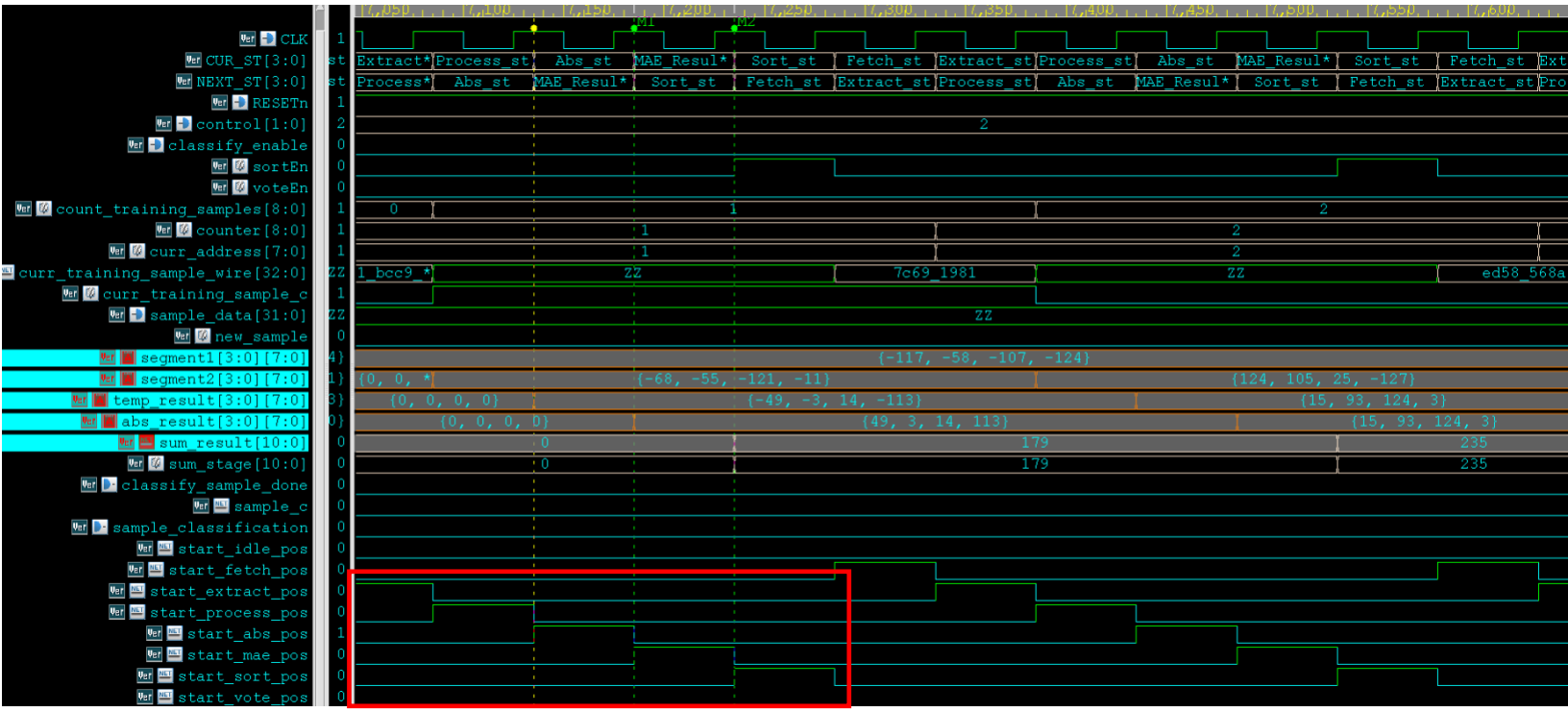
- PE2 and PE3 classified their current sample as ‘1’, therefore counter_res_2 and counter_res_3 were updated.
- Sum of counter_res_1 and counter_res_2 is saved in sum_1.
- Sum of counter_res_3 and counter_res_4 is saved in sum_2.
- sum_3 holds the sum of sum_1 and sum_2.
- Threshold of the KNN core is 250, hence when sum_3 reaches 250, the KNN_done and the KNN_classification (classification is ‘1’) signals rise to ‘1’.
- Time between the beginning of the first fetch state and the beginning of the done state : 1,971,200ns.
- Latency : 1,971,200/1024 = 1925ns.
- Measurements match the calculations in section 3.7.2 .

KNN PE:

Extraction phase during the first iteration:

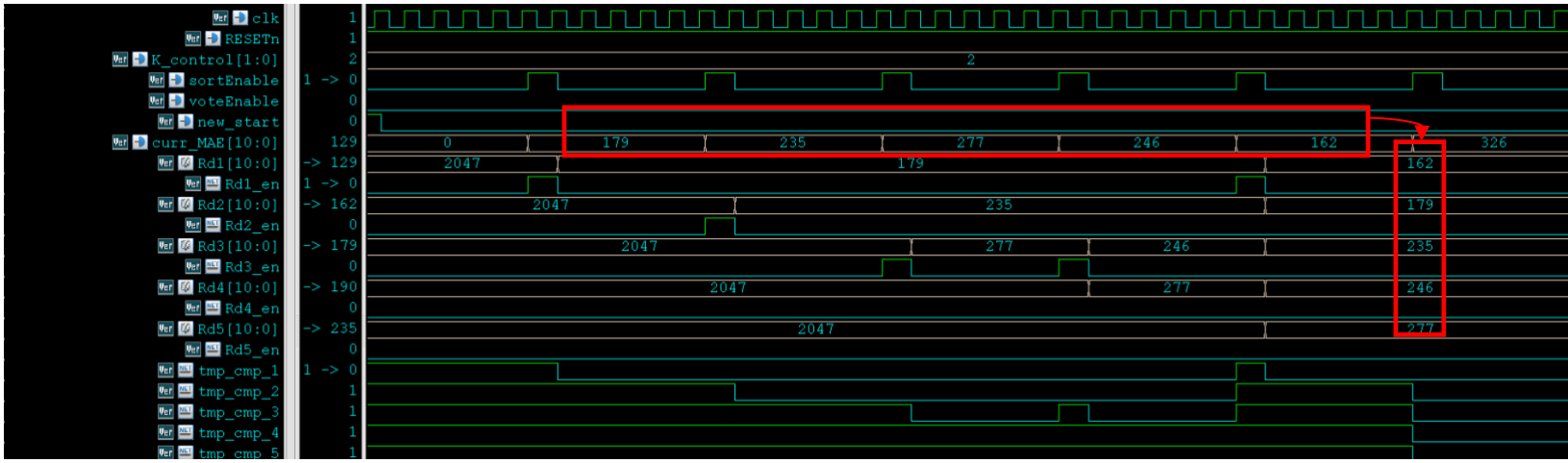
- curr_training_sample_wire [31:0] is separated into 4 8-bit signals that make segment2.
- The 33rd bit of curr_training_sample_wire is saved in the curr_training_sample_c signal.
- Sample_data [31:0] is separated into 4 8-bit signals that make segment1.

Calculation phase inside the KNN PE :

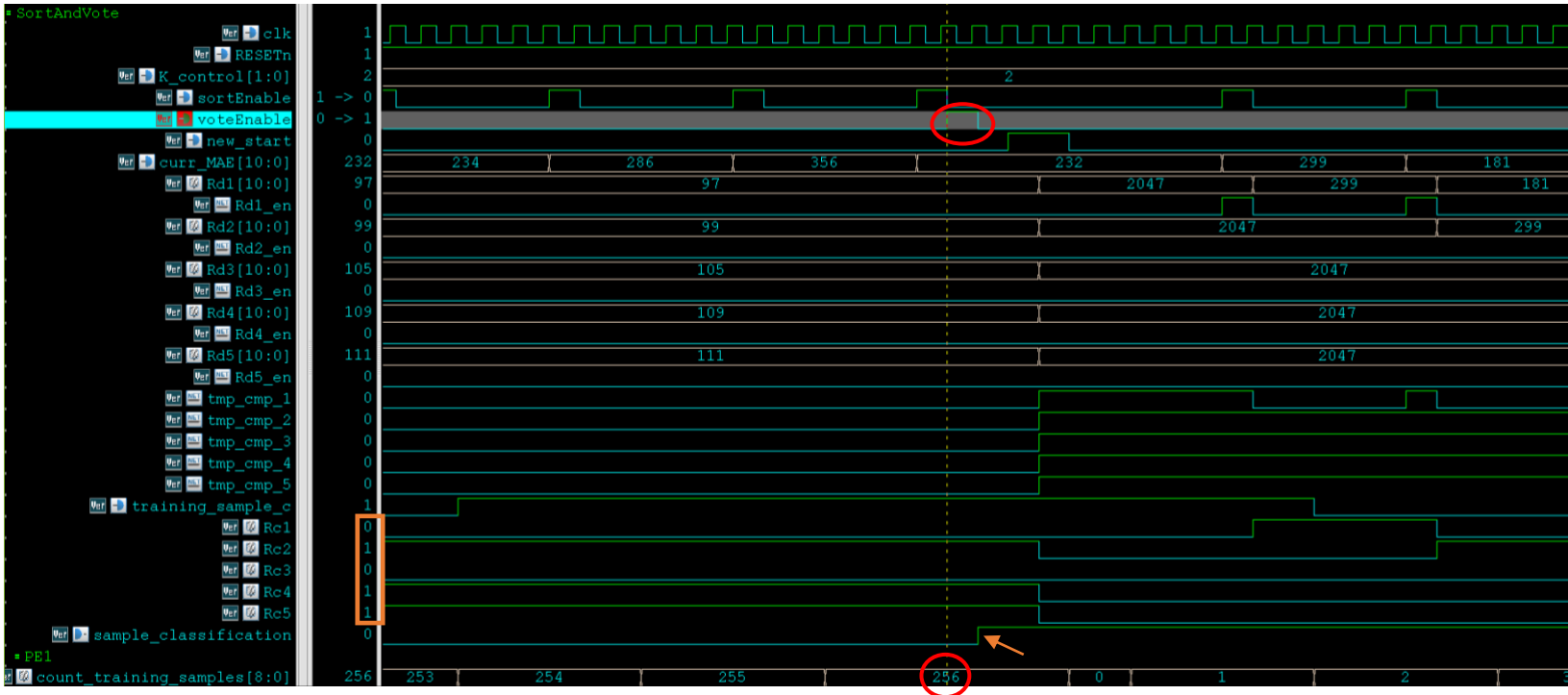


- segment1 – segment2 = temp_result
- abs_result = abs_value(temp_result)
- sum_result = sum(abs_result)
- Each one of the states of the state machine takes one cycle.

Sort & vote phase :

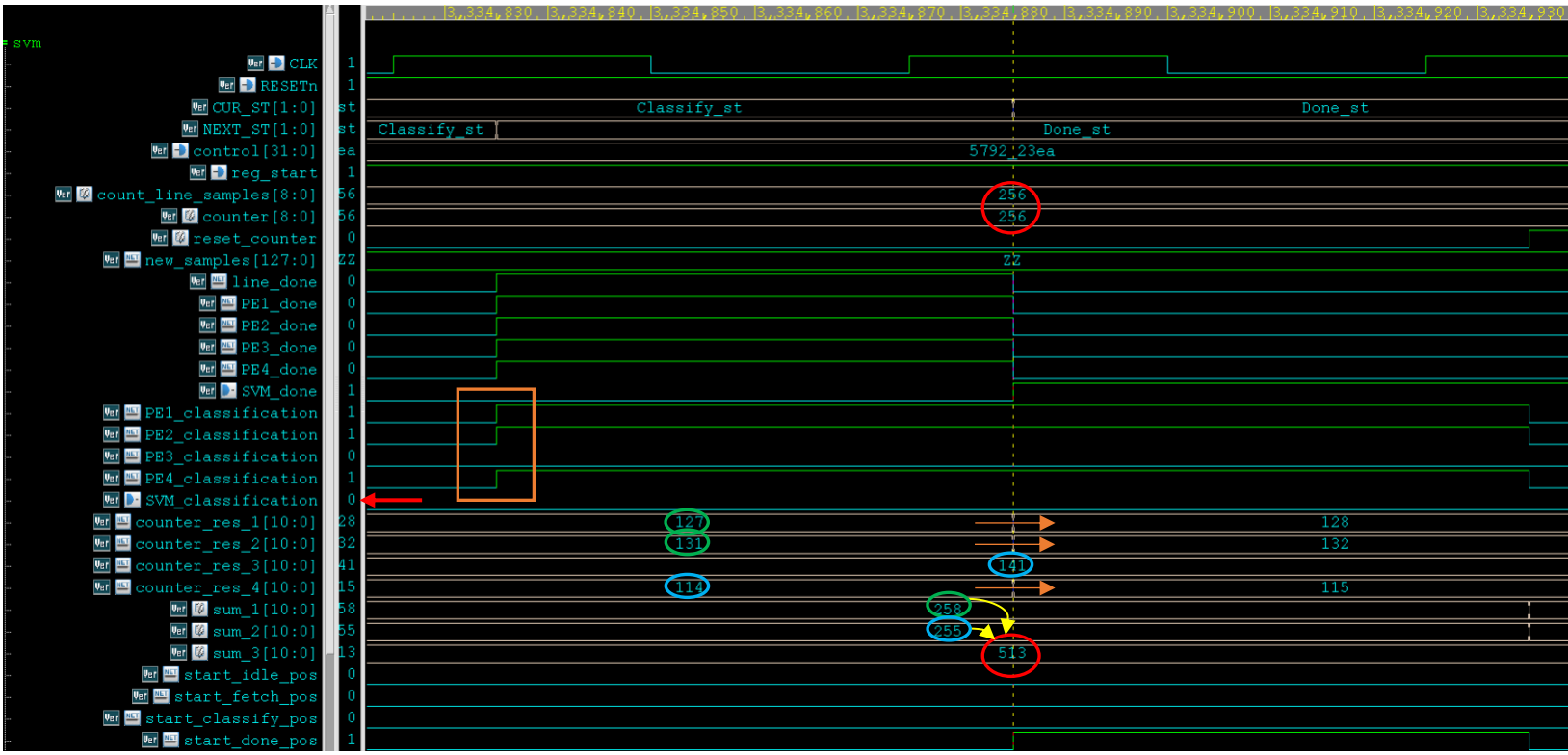


Newly calculated MAEs are shifted in order to store them sorted in the registers Rd 1-5 (in depth explanation in section 3.6.1).



- Once the counter reaches 256, indicating the last training sample, the ‘vote’ signal rises in order to vote on the classification of the current sample according to the five classifications with the 5 smallest MAEs (5 nearest neighbors).
- The 5 samples with the smallest distances have a majority (3-2) of ‘1’ classification, hence the classification result of the current sample is ‘1’.

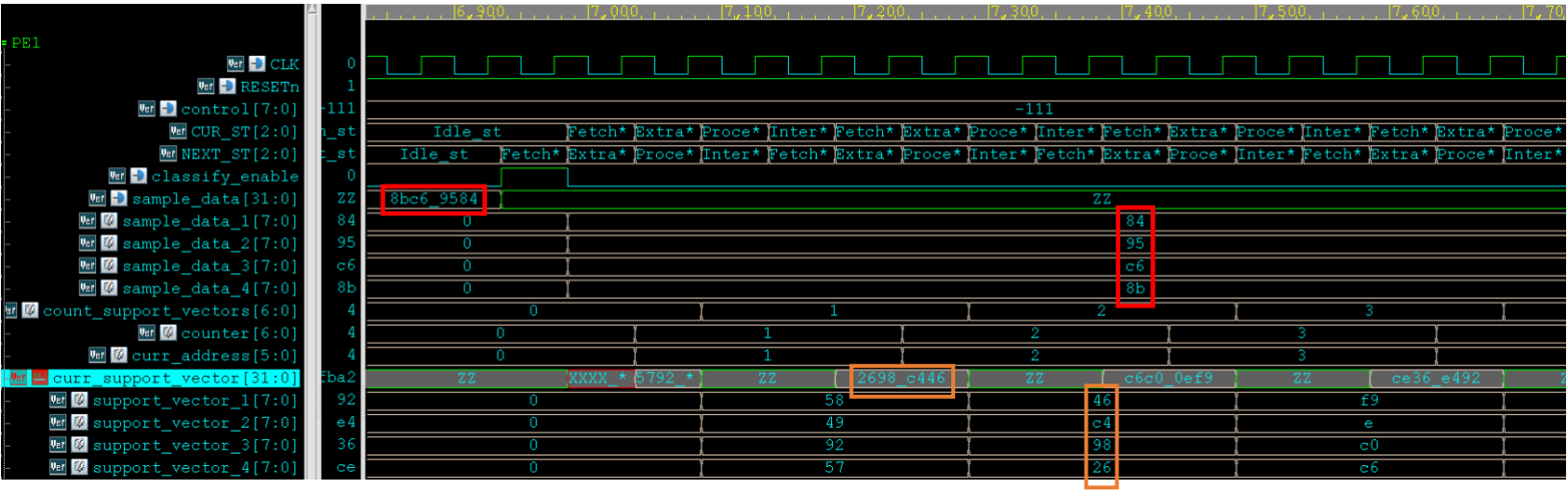
SVM Core:



- PE1, PE2 and PE4 classified their current sample as '1', therefore counter_res_1, counter_res_2 and counter_res_4 were updated.
- Sum of counter_res_1 and counter_res_2 is saved in sum_1.
- Sum of counter_res_3 and counter_res_4 is saved in sum_2.
- sum_3 holds the sum of sum_1 and sum_2.
- Threshold of the SVM core is 700, and sum_3 reaches a maximum of 513 , therefore the SVM_classification signals rise to stays at '0'.
- Time between the beginning of the first fetch state and the beginning of the done state : 332,800ns.
- Latency : 332,800ns/1024 = 325ns.
- Measurements match the calculations in section 3.7.2 .

SVM PE:

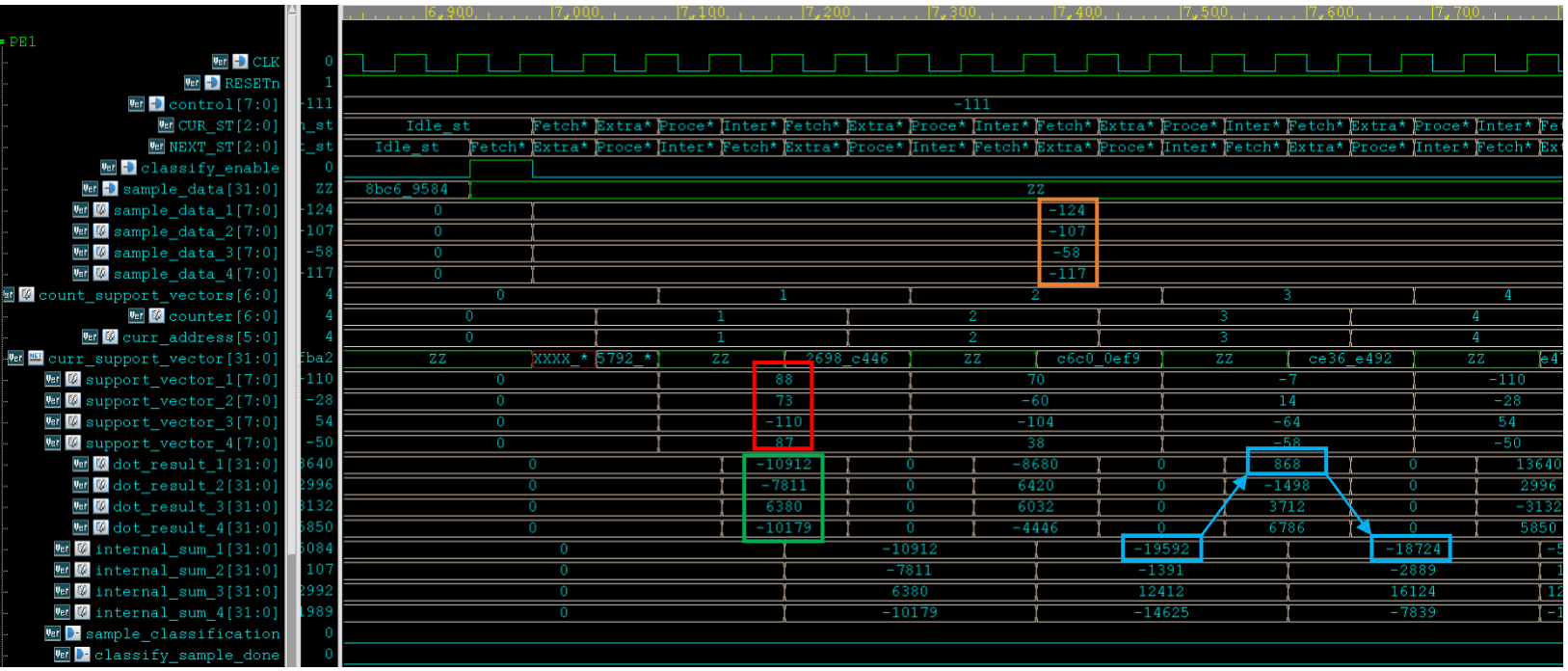
Extraction phase during the first iteration:



- sample_data [31:0] is separated into 4 8-bit signals sample_data1-4.
- curr_support_vector [31:0] is separated into 4 8-bit signals support_vector1-4.

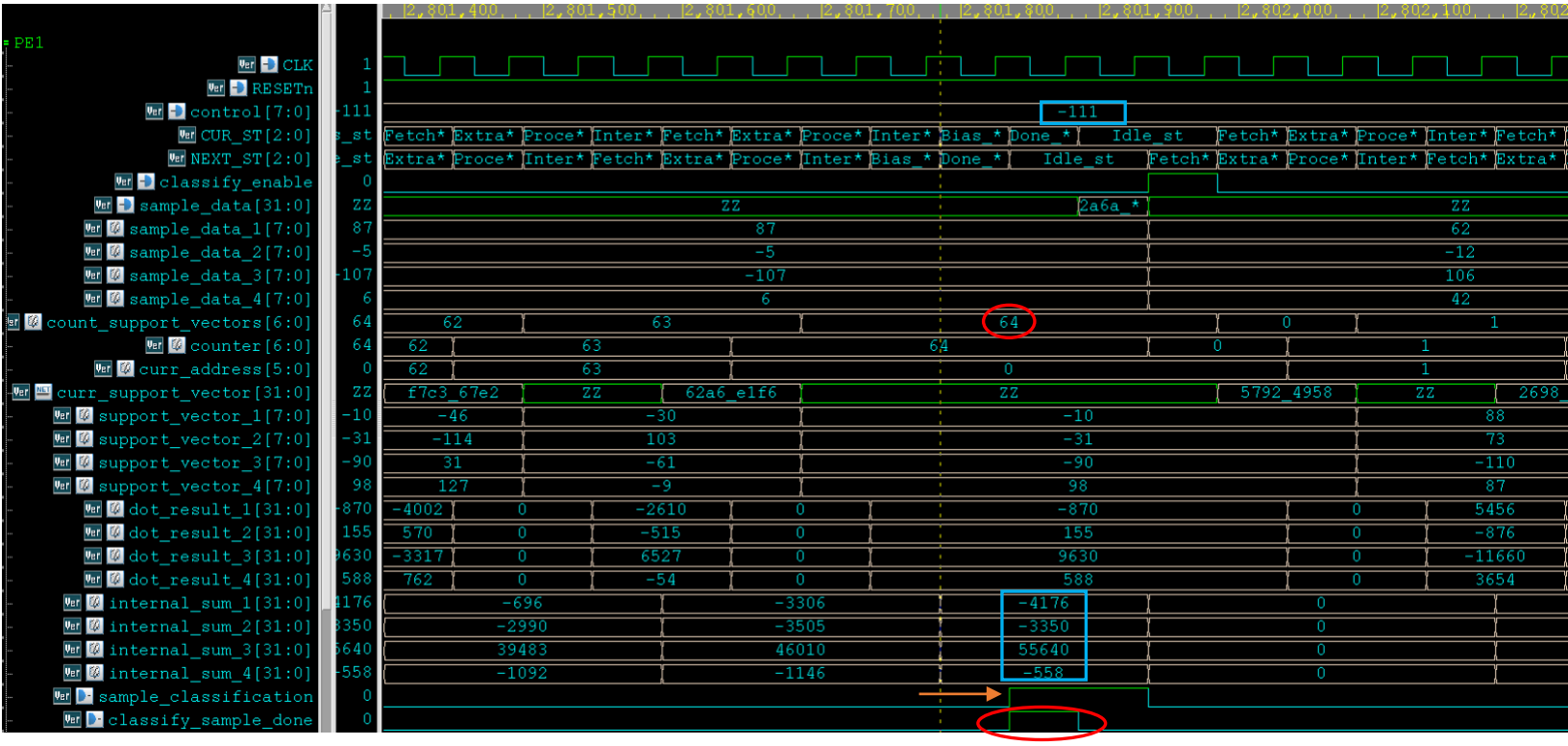
Calculation Phase:

Dot result and internal sum:



- sample_data_i * support_vector_i = dot_result_i (1 <= i <= 4).
- internal_sum_i += dot_result_i

Adding the bias and comparing to zero:

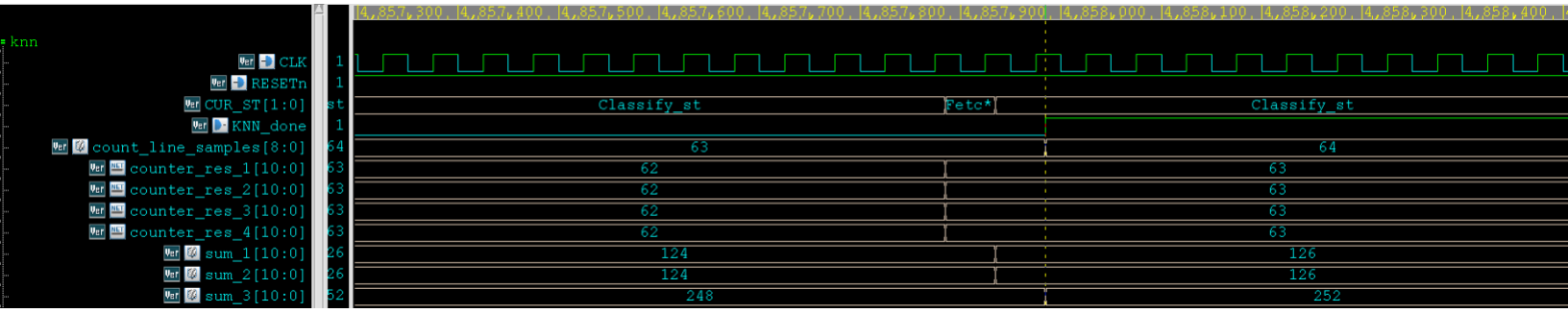


- After going over all of the 64 support vectors and count_support_vectors reaches 64, classify_sample_done signal rises signaling the end of the classification of the current sample.
- The sum of internal_sum_1, internal_sum_2, internal_sum_3, internal_sum_4 and the bias is calculated and compared to zero, since the result is positive (47,445), the sample_classification is '1'.

KNN edge cases:

All training data of the KNN core are classified zero:

All training data of the KNN core are classified one:



Threshold is 250, thus when sum_3 becomes 252, KNN_done rises signaling the end of the classification process (classification is ‘1’).

4 Summery:

4.1 Project's summary:

The project's goal is to design and implement an optimized hardware accelerator using KNN and SVM-based applications, utilizing the APB Protocol for the interface. Focused on inference, the accelerator employs SVM and KNN classifiers, assuming pre-training.

We started the project journey with a comprehensive introduction, followed by self-learning about SVM and KNN algorithms and studying the APB protocol. The research paper, "Embedded Low-Power Processor for Personalized Stress Detection", served as a pivotal reference in our planning phase.

The planning phase involved structuring the top-level architecture based on a Block Diagram from the referenced research paper. Subsequently, optimization efforts led to the segmentation of cores, each housing 4 processing elements for parallelized operation, with each unit handling 0.25 of the data and some parameters were made user-customizable, ensuring flexibility as discussed before.

In the next step, we theorized performance diagrams for each component, outlining expected throughput and latency. Once planning was complete, the implementation phase began with the register file, followed by individual module implementations and interconnection.

Zero order verification and testing, including simulation, validated the project's functionality.

The final step incorporated the synthesis part, where one of the RAMs, of the SVM PE, transitioned to real memory.

Overall, the project demonstrates a systematic approach from learning and planning to implementation, with a focus on optimizing hardware performance for stress detection applications.

4.2 Future work:

These ideas provide a starting point for future work, offering avenues for expansion, optimization, and diversification of this project:

1. Enhanced Hardware Optimization: Explore additional hardware optimization techniques to further enhance the performance of the accelerator. This may include refining the design of processing elements, improving parallelization strategies, or implementing more advanced algorithms for increased efficiency.

2. Real-Time Training Support: Extend the functionality to include real-time training capabilities. This would involve designing hardware components that can handle the training phase of machine learning models, enabling the accelerator to adapt and learn from new data in real-time.

3. Cross-Domain Applications: Investigate the adaptability of the accelerator for cross-domain applications. Explore how the hardware can be configured and optimized for diverse fields such as healthcare, finance, or environmental monitoring, expanding its utility beyond stress detection.

4. Benchmarking and Performance Evaluation: Conduct extensive benchmarking and performance evaluations across a variety of datasets and stress detection scenarios. Compare the accelerator's performance against existing solutions and assess its scalability, robustness, and generalization capabilities.

5. Clock Domain Separation: Implement an additional clock for the core processing units, independent of the APB interface clock. This separation aims to enhance system performance by allowing internal operations to run at a faster, independent frequency. Evaluate the impact on data transfer rates, overall efficiency, and explore dynamic clock frequency adjustments to adapt to varying workloads.

5 References

[1] K-Nearest Neighbors Algorithm - IBM:
https://www.ibm.com/topics/knn
[2] K-Nearest Neighbors Algorithm – wikipedia:
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#Algorithm
[3] A guide to the types of machine learning algorithms and their applications:
https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html

[4] Hamoud Younes, Ali Ibrahim, Mostafa Rizk, Maurizio Valle. An Efficient Selection-Based kNN Ar- chitecture for Smart Embedded Hardware Accelerators. IEEE Open Journal of Circuits and Systems, 2021, 2, pp.534-545. 10.1109/OJCAS.2021.3108835 . hal-03668082 :
https://hal.science/hal-03668082/document

[5] Verilog Priority Encoder:
https://www.geeksforgeeks.org/verilog-priority-encoder/
[6] RTL Combinational Circuit - Design Examples - Barrel Shifter RTL Combinational Circuit:
https://community.element14.com/technologies/fpga-group/b/blog/posts/systemverilog-study-notes-barrel-shifter-rtl-combinational-circuit
[7] Support Vector Machine Algorithm:
https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm
[8] K-Nearest Neighbor(KNN) Algorithm for Machine Learning
https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106