An Efficient Digital VLSI Implementation of Gaussian Mixture Models-Based Classifier

Minghua Shi and Amine Bermak, Senior Member, IEEE

Abstract—Gaussian mixture models (GMM)-based classifiers have shown increased attention in many pattern recognition applications. Improved performances have been demonstrated in many applications, but using such classifiers can require large storage and complex processing units due to exponential calculations and a large number of coefficients involved. This poses a serious problem for portable real-time pattern recognition applications. In this paper, first the performance of GMM and its hardware complexity are analyzed and compared with a number of benchmark algorithms. Next, an efficient digital hardware implementation is proposed. A number of design strategies are proposed in order to achieve the best possible tradeoffs between circuit complexity and real-time processing. First, a serial-parallel vector-matrix multiplier combined with an efficient pipelining technique is used. A novel exponential calculation circuit based on a linear piecewise approximation is proposed to reduce hardware complexity. The precision requirement of the GMM parameters in our classifier are also studied for various classification problems. The proposed hardware implementation features programmability and flexibility offering the possibility to use the proposed architecture for different applications with different topologies and precision requirements. To validate the proposed approach, a prototype was implemented in 0.25-μm CMOS technology and its operation was successfully tested for gas identification application.

Index Terms—Digital VLSI architecture, GMM, pattern recognition, reconfigurable architecture.

I. INTRODUCTION

The Gaussian mixture models (GMM) classifier has gained increasing attention in the pattern recognition community. GMM can be classified as a semi-parametric density estimation method since it defines a very general class of functional forms for the density model. In this mixture model, a probability density function is expressed as a linear combination of basis functions. Improved classification performances have been demonstrated in many pattern recognition applications [1]–[10]. Performance figures of more than 95% have already been reported for applications such as electronic nose [11] and gas identification [2]. Another interesting property of GMM is that the training procedure is done independently for each class in turn by constructing a Gaussian mixture of a given class. Adding a new class to a classification problem does not require retraining the whole system and does not affect the topology of the classifier making it attractive for pattern recognition applications. While GMM provides very good performances and interesting properties as a classifier, it presents some problems that may limit its practical use in real-time applications. One problem is that GMM can require large amounts of memory to store various coefficients and can require complex computations mainly involving exponential calculations. Thus, this scheme can be put to efficient practical use only if good hardware implementation strategies are developed. In this paper, we propose an efficient digital VLSI implementation that we believe can meet the computational requirement of GMM-based classifiers. First, after analyzing the complexity of the GMM classifier it was found that the vector-matrix multiplication and the exponential calculations are the most critical operations in the classifier. A good tradeoff between real-time processing and hardware resources requirements is obtained using a serial-parallel architecture and an efficient pipelining strategy. Second, a linear piecewise function (LPF) is proposed to replace the exponential calculation. Implementing LPF-based GMM, also permits to avoid the need for using area consuming look-up table (generally used in digital implementation) to implement the exponential function. The effect of both limited precision and the mixture models approximation using LPF on the classification performance is investigated using seven different data-sets. These data-sets are also used to compare the performance of GMM with other benchmark classifiers. A proof-of-concept prototype was implemented in a 0.25-μm CMOS process using automatic placement and routing design methodology. The design was made flexible and programmable making it a general purpose processor which can be applied to different classification problems.

The paper is organized as follows. Section II introduces the GMM algorithm and compares its performance with benchmark classifiers for seven different classification data-sets. Section III analyzes the complexity of GMM classifier and compares its computational requirements with some benchmark classifiers. Important hardware considerations are also discussed in Section III such as the required precision and the effect of using LPF approximation on the classification performance. Section IV presents the digital VLSI implementation of the GMM classifier and reports the performance of the chip for gas identification. Finally, Section V concludes this paper.

II. ALGORITHM CONSIDERATIONS

The task of a pattern recognition algorithm is to set a decision rule, which optimally partitions the data space into regions, one for each class $C_k$. A pattern classifier generates a class label for an unknown feature vector $x \in \mathbb{R}^d$ from a discrete set of previously learned classes. One way to build a classifier...
is to estimate the class-conditional densities by using representation models for how each pattern class populates the feature space. In this approach, classifier systems are built by considering each class in turn, and estimating the corresponding class-conditional densities $\psi(x|C_k)$ from the data. The $K$ nearest neighbor (KNN) is classified as a nonparametric density estimation, which has been widely used for classification. Despite its simplicity, KNN often performs very well and is an important benchmark method. However, one drawback of KNN is that all the training data must be stored, and a large amount of processing is needed to evaluate the density for a new input pattern. An alternative method is to combine the advantages of both parametric and nonparametric methods, by allowing a very general class of functional forms in which the number of adaptive parameters can be increased to build more flexible models. This leads us to a powerful technique for density estimation, called mixture models [12]. In a GMM, a classifier can be constructed by evaluating the posterior probability of an unknown input pattern $x$ belonging to a given class $C_k$ expressed as $\psi(C_k|x)$. Based on Bayes’ theorem, $\psi(C_k|x)$ can be written as

$$\psi(C_k|x) = \frac{\psi(C_k)\psi(x|C_k)}{\psi(x)} \quad (1)$$

where $\psi(C_k)$ is the frequency of a given training sample in the data-set and the unconditional density $\psi(x)$ is given by

$$\psi(x) = \sum_{k=1}^{c} \psi(x|C_k)\psi(C_k) \quad (2)$$

which ensures that the posterior probabilities sum-up to unity. In the GMM case, the class conditional densities $\psi(x|C_k)$ can be expressed as a linear combination of basis functions $\psi(x|j)$. A model with $M$ components is described as a mixture distribution [12]

$$\psi(x|j) = \sum_{j=1}^{M} \psi(j)\psi(x|j) \quad (3)$$

where $\psi(j)$ are the mixing coefficients of the component density functions $\psi(x|j)$. Each mixture component is defined by a Gaussian parametric distribution in $d$ dimensional space

$$\psi(x|j) = \frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j) \right\}. \quad (4)$$

The parameters to be estimated are the mixing coefficients $\psi(j)$, the covariance matrix $\Sigma_j$, and the mean vector $\mu_j$. The form of the covariance matrix can be spherical, diagonal, or full. Maximizing the data likelihood is often used as a training procedure for mixture models. The log likelihood of the data-set $(x_1, \ldots, x_n)$, which is treated as an error, is defined by

$$l = \sum_{i=1}^{n} \log \psi(x_i). \quad (5)$$

The expectation-maximization (EM) algorithm, a well-established and common technique, is used for maximum-likelihood parameter estimation [13]-[16]. The EM algorithm iteratively modifies the model parameters starting from the initial iteration $k = 0$. EM guarantees a monotonically nondecreasing likelihood, although its ability to find a local maximum depends on parameter initialization. For GMM, the EM optimization can be carried out analytically with a simple set of equations [13], where the mixing coefficients are estimated by

$$\psi^{k+1}(j) = \frac{1}{n} \sum_{i=1}^{n} \psi^k(j|x_i) \quad (6)$$

and the estimate for the means for each component is given by

$$\mu_j^{k+1} = \frac{\sum_{i=1}^{n} \psi^k(j|x_i)x_i}{\sum_{i=1}^{n} \psi^k(j|x_i)} \quad (7)$$

and, finally, the update equation for the full covariance matrix is

$$\Sigma_j^{k+1} = \frac{\sum_{i=1}^{n} \psi^k(j|x_i)(x_i - \mu(k+1)_j)(x_i - \mu_j^{k+1})^T}{\sum_{i=1}^{n} \psi^k(j|x_i)}. \quad (8)$$

The minimum description length (MDL) is applied to estimate the number of density models of each class [13]. GMM can approximate any continuous density with an arbitrary accuracy provided the model has a sufficiently large number of components, and provided the parameters of the model are chosen correctly. The training process can be computationally intensive compared to the simple procedure needed for parametric methods, however, the recall or generalization procedure is quite simple and can be performed very rapidly when compared to KNN. In addition, the GMM training procedure is performed independently for each class in turn by constructing a Gaussian mixture of a given class. Adding a new class to a given classification problem does not require retraining the whole system and does not affect the topology of the classifier.

The mixture models are built on the feature space and $\psi(C_k|x)$ can be calculated using (1). The probability of misclassification is minimized by selecting the class $C_k$ having the largest posterior probability. A feature vector $x$ is assigned to class $C_k$ if

$$\psi(C_k|x) > \psi(C_j|x) \quad \forall j \neq k. \quad (9)$$

Since the unconditional density $\psi(x)$ is independent of the class, it may be omitted from the Bayes formula as the classification process consists of comparing the posterior probabilities. Thus, we can write the criterion in the newly simplified form as

$$\psi(x|C_k)\psi(C_k) > \psi(x|C_j)\psi(C_j) \quad \forall j \neq k. \quad (10)$$

As a consequence, a GMM classifier can be seen as providing a rule for assigning each point in the feature space to one class out of $c$ classes [13]. The feature space is, therefore, partitioned into $c$ decision regions and $\psi(x|C_k)\psi(C_k)$ is used as a decision boundary during the evaluation phase.
TABLE I
DATA-SET USED FOR TESTING THE PERFORMANCE OF GMM CLASSIFIER.
THE TWO LAST ODOR DATA-SETS WERE OBTAINED USING OUR
OWN EXPERIMENTAL SETUP. THESE ARE RELATED TO GAS
DISCRIMINATION APPLICATION

<table>
<thead>
<tr>
<th>Data-sets</th>
<th># of samples</th>
<th># of variables</th>
<th># of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis</td>
<td>80</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Crabs</td>
<td>200</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Breast</td>
<td>226</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Vehicle</td>
<td>846</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>Iris</td>
<td>150</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Odor 1</td>
<td>86</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Odor 2</td>
<td>220</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

therefore, used to find the decision boundaries in the evaluation period, which can be expressed as

\[ \psi(x|C_k) \psi(C_k) = \psi(C_k) \sum_{j=1}^{M} \psi(j) \psi(x|j). \]  

To evaluate the performance of GMM as a classifier, we performed discrimination experiments on seven data-sets. Table I describes the seven data-sets and illustrates the number of samples, variables, and classes used. The breast cancer data-sets are obtained from the University of Wisconsin Hospitals, Madison. The Hepatitis, Vehicle, and Iris data-sets are taken from the UCI repository. The crabs data-set is adopted from Campbell & Mahon (1974) on the morphology of rock crabs of genus Leptograpsus. The last two data-sets are related to gas discrimination and were obtained from our gas sensor experimental setup [18]. The first Odor data-set is based on the detection of three gases (CO, H₂, mixture of CO, and H₂) using seven tin-oxide gas sensors. The second Odor data-set is based on the detection of five gases (CO, CH₄, H₂, mixture of CO, and CH₄, mixture of CO and H₂) using eight tin-oxide gas sensors.

We compared GMM with a wide range of classification algorithms including multilayer perceptron (MLP), KNN, radial basis functions (RBF), and probabilistic principal component analysis (PPCA) [13]. Different competing classifiers are built on the previously described data-sets (Table I). Generalization performances were estimated using the ten-fold cross-validation approach. In this approach, the samples are divided into ten groups and at each step one group is selected for test while the remaining groups are used for training the classifiers. The procedure is repeated for the whole set and the average performance is calculated. Table II reports the classification performance using different classifiers based on MatLab simulation results. For GMM, the simulation results are reported for full (GMM-F) and diagonal (GMM-D) covariance matrices. For our specific data-sets, it can be shown that the performance of GMM-F is on average 10% higher as compared to the one of GMM-D. Nevertheless, one should note that using diagonal covariance matrix results in reduced complexity as compared to full covariance matrix and, therefore, studying the tradeoff in terms of performance and complexity could be very interesting to explore further.

III. FROM GMM ALGORITHM TO A HARDWARE-FRIENDLY
ARCHITECTURE

Classification performance is not the only criteria to be considered when implementing classifiers. Hardware complexity as well as the required accuracy must be studied very carefully for the problem at hand. For the sake of flexibility and easy storage of various GMM coefficients, we selected the digital technology over its analog counterpart.

A. Complexity Study

One important task when considering GMM hardware complexity is to estimate the number of memory units storing the parameters of the Gaussian models. In order to reduce the memory size, new set of parameters (constant \( K_j \) and a triangular matrix \( G_j \)) are defined and used instead of \( \psi(x|C_k), \psi(j), \Sigma_j^{1/2}, \) and \( \Sigma_j^{-1}(\Sigma_j^{-1}) \) (a full matrix). The new coefficients \( K_j \) and \( G_j \) are given by

\[ K_j = \frac{\psi(C_k)\psi(j)}{(2\pi)^{d/2} \Sigma_k^{1/2}} \]  

\[ G_j^+G_j = 2^{-1} \Sigma_j^{-1}. \]  

\( G_j \) is a triangular matrix introduced in order to reduce the complexity as compared to dealing with a full matrix when it comes to (4) calculation. If we assume that

\[ z_j = \left( x - \mu_j \right)^+ G_j \left( x - \mu_j \right)^+ G_j^+ \]  

equation (11) can be rewritten as

\[ \psi(x|C_k) \psi(C_k) = \sum_{j=1}^{M} K_j \exp\{-z_j\}. \]  

Thus, the calculation of (15) can be divided into three steps: evaluation of parameter \( z_j \), the exponential calculation, and the multiplication with constant \( K_j \). It can be noted that the evaluation of parameter \( z_j \) is the most demanding operation. Using (15), we can reduce the complexity of GMM. In order to assess GMM computational requirements, Fig. 1 shows the GMM data flow block diagram. First, \( s = x - \mu_j \) is evaluated after \( d \) subtractions (\( d \) is the dimension of \( x \) vector). Next, \( y_j = (x - \mu_j)^+ G_j \) is calculated using a serial-parallel vector-matrix multiplier. A serial-parallel approach is selected because it permits to obtain the best tradeoff in terms of speed and hardware requirements. In addition, providing the test patterns \( x \) in a parallel fashion will be prohibitively expensive in terms of

TABLE II
CLASSIFICATION PERFORMANCE OF DIFFERENT CLASSIFIERS BASED ON THE SEVEN DATA-SETS. TEN-FOLD CROSS-VALIDATION APPROACH IS USED TO EVALUATE THE CLASSIFICATION ACCURACY (IN %). GMM-F AND GMM-D STANDS FOR GMM CLASSIFIER USING FULL AND DIAGONAL COVARIANCE MATRICES, RESPECTIVELY

<table>
<thead>
<tr>
<th>Data-sets</th>
<th>GMM-F</th>
<th>GMM-D</th>
<th>MLP</th>
<th>KNN</th>
<th>RBF</th>
<th>PPCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis</td>
<td>87.50</td>
<td>85</td>
<td>83.75</td>
<td>81.25</td>
<td>83.75</td>
<td>75</td>
</tr>
<tr>
<td>Crabs</td>
<td>100</td>
<td>88.50</td>
<td>100</td>
<td>96.50</td>
<td>100</td>
<td>99.50</td>
</tr>
<tr>
<td>Breast</td>
<td>93.20</td>
<td>90.90</td>
<td>91.36</td>
<td>92.30</td>
<td>92.27</td>
<td>92.75</td>
</tr>
<tr>
<td>Vehicle</td>
<td>84.05</td>
<td>72.50</td>
<td>65.59</td>
<td>65.24</td>
<td>65.48</td>
<td>45.95</td>
</tr>
<tr>
<td>Iris</td>
<td>98</td>
<td>95.30</td>
<td>98</td>
<td>96</td>
<td>95.33</td>
<td>98</td>
</tr>
<tr>
<td>Odor 1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>98.75</td>
<td>96.25</td>
<td>97.50</td>
</tr>
<tr>
<td>Odor 2</td>
<td>94.10</td>
<td>85.91</td>
<td>93.75</td>
<td>92.50</td>
<td>86.80</td>
<td>84.10</td>
</tr>
</tbody>
</table>
input/output (I/O) requirements. The $G_j$ is a triangular matrix, so the calculation of $y_j$ needs $d(d + 1)/2$ multiplications and $(d - 1)d/2$ additions. $z_j = y_j y_j^T$ requires $d$ multiplications and $(d - 1)$ additions in the square unit and accumulator. The above calculations are repeated $M$ times to obtain $\phi(x|C_k)_k(x|C_k)$. In addition, GMM also requires a number of memory units in order to store its coefficients such as $K$, $G$, and $\mu$.

The complexity of the GMM classifier is quadratically dependent upon the dimension of the Gaussian models. One way to reduce GMM complexity is to project data to a lower dimensional space using one of the well-established techniques such as the Principal Components Analysis (PCA), which attempts to preserve most of the information in the original data space while projecting to a lower dimension. To evaluate the impact of introducing such a preprocessing technique on GMM classification accuracy, we calculated again classification accuracy for all possible dimensional projections $p = d \rightarrow 2$. For example, in the case of the Odor 2 data-set, which includes eight variables, the classification performance was evaluated using $p = 8 \rightarrow 2$ principal components. Table III shows the classification performance of GMM as a function of the number of principal components. It is found that for all our tested data-sets, except the Vehicle data-set, it is possible to reduce the dimension and, hence, the complexity of GMM while maintaining the classification performance to a reasonable level. Five principal components are sufficient for both Hepatitis and Odor 2 data-sets, while only three principal components are sufficient for Crabs, Breast, Iris, and Odor 1 data-sets. A significant saving in terms of computational requirement is clearly achieved, particularly for the case of hepatitis data-sets as only five input variables can be used instead of the original 19. However, 16 principal components are necessary to achieve the same performance, when compared to the original data-set (18 input variables for the Vehicle data-set).

The computational requirements of GMM were also compared with MLP, KNN, RBF, and PPCA. The complexity measures are based on PCA projection for the seven data-sets. Fig. 2 reports the average computational requirements for the seven data-sets in terms of number of addition, multiplication (MUL), nonlinear functions (Exp) as well as memory units noted as MU. The number of MU is dependent upon the selected precision or the precision requirement. The numbers reported correspond to the number of words coefficients required for each classifier. If for example an 8-bit precision ($b = 8$) is used the number reported for MU needs to be multiplied by 8.

From Fig. 2, we can deduce that KNN presents the largest computational and memory requirements while MLP classifier has on average the lowest complexity and the smallest memory requirements. It is, however, important to note that the GMM classifier has the best classification performance and yet acceptable complexity and memory requirements. The main advantage of GMM is related to its class-based training. In other words, the topology of the overall system does not require major alterations when dealing with different applications. In addition, adding or removing a new class to a given data-set is done without retraining the whole system and without changing the classifier structure, which gives a greater flexibility to the classifier. Both MLP and RBF require complete modification of the network structure and retraining the whole system when adding a new class to the classification problem. In addition, designing programmable network topology may
not be an easy task to deal with hardware reconfigurability requirement for MLP and RBF. For the KNN classifier, the complexity and memory requirement are proportional to the number of data patterns. Adding a new class will increase its complexity and memory requirement very rapidly. The GMM classifier presents similar characteristics compared to KNN. The structure and parameters of the original system will not change when adding a new class. New parameters of the Gaussian models related to the new class need to be added into the existing system to classify a newly added class. However, compared to KNN, the increase of complexity and memory requirement is limited for GMM (the number of Gaussian models is much smaller than the number of data patterns).

B. What Is the Effect of LPF-Based Approximation?

One of the major challenges when implementing the GMM classifier is related to the complexity of the exponential function calculation. In a digital implementation, the lookup table (LUT) can be used to implement the exponential function by means of discrete values approximation. Unfortunately, LUT-based implementation suffers from large on-chip storage requirement, especially when using high precisions. Let us consider the example of the Odor 2 data-set and evaluate the size of the LUT for this example. For our implementation, the word length of the inputs to the exponential unit shown in Fig. 1 is 40 bits. If the exponential unit is implemented using LUT, 21 bits are needed in order to cover the input range and a 16-bit precision is required for the exponential results (output of the LUT). Given these requirements, the size of the LUT will be 4 MB ($2^{21} \times 16$ bits). It is very clear that this requirement is very area and time consuming as accessing a large memory will also affect the speed of the GMM processor. Taylor series is another alternative that can be used to approximate the exponential function ($\exp(-z)$). However the main issue related to the use of Taylor series is the need for square and cubic calculations, which require extra area consuming multipliers. Furthermore, Taylor series approximation is only valid when $z < 1$. This will probably require normalizing the input prior to applying the approximation, which will introduce further complexity to the overall system. We, therefore, propose to approximate the exponential function by using linear piecewise function (LPF). LPF is proven to be hardware-friendly [19], [20] and presents much lower storage requirement as compared to the LUT-based exponential function. However, the effect of the mixture model using LPF instead of the exponential function needs to be thoroughly investigated. In our experiment, three possible piecewise implementations are explored as illustrated by the three functions $f_1(z)$, $f_2(z)$, and $f_3(z)$ with $z \geq 0$ (Fig. 3).

As a result of this approximation, the shape of the Gaussian models will be altered. Fig. 4 shows the shape of the mixture models in 1-D using LPF compared with that of the Gaussian mixture models. Obviously, the shape of the new mixture model is closer to the Gaussian model when using a higher number of pieces in the piecewise function

$$f_1(z) = \begin{cases} 1, & \text{if } z < a \\ 0, & \text{if } z \geq a \end{cases}$$ (16)
The solid vertical line is the decision boundary of the GMM classifier using exponential function and the dashed vertical line is the decision boundary of the GMM classifier using LPF approximation.

\[
f_2(z) = \begin{cases} 
1, & \text{if } z < a \\
-k_1(z - b), & \text{if } a \leq z < b \\
0, & \text{if } z \geq b 
\end{cases} \quad (17)
\]

\[
f_3(z) = \begin{cases} 
1, & \text{if } z < a \\
-k_1(z - b'), & \text{if } a \leq z < b \\
-k_2(z - c), & \text{if } b \leq z < c \\
0, & \text{if } z \geq c
\end{cases} \quad (18)
\]

However, one can note that the effect of such an approximation will have minimum effect if the mismatch between the two decision boundaries is kept minimum. Our objective is not to propose a new classifier but to approximate the trained GMM classifier using a hardware-friendly LPF-based GMM, so that the performance mismatch is minimized. In order to do so, we propose to first build the original GMM, which would permit to find the main parameters such as the number of components, the mean vector, and the covariance matrix. Once this is achieved, the LPF parameters are found based on the same training data-set and using the performance criteria as the optimization parameter. While the LPF-based GMM is no longer a Gaussian mixture, which may not satisfy some mathematical properties, however, the main parameters of GMM-based classifier are inherently kept since the LPF is based on the same mean vector (\(\mu\)) and the same covariance matrix (\(\Sigma\)). The effect of using LPF approximation is illustrated in Fig. 5. We can note that the key point here is to keep the mismatch between the two decision boundaries to a minimum level. In the example shown in Fig. 5, the decision boundary mismatch between the GMM and LPF-based GMM is not very large mainly because the mean and the covariance matrix are kept unchanged. One should also note that this mismatch is reduced when using higher order LPF and optimizing a certain cost function (performance mismatch in our case) during the training phase.

The process of building the classifier using LPF can be divided into two major phases: the training and the test procedure. The parameters of the classifier are selected during the training phase. We first obtain the parameters of the mixture models such as \(\mu\) and \(\Sigma\) based on the training data-set. The same training data are fed into the classifier in which the exponential function is replaced by LPF, while the other parameters of the Gaussian model such as the mean vector and the covariance matrix are kept unchanged. Next, the parameters of LPF are optimized such that the mismatch between decision boundaries in the LPF-based GMM and the original GMM is minimized. This process can be shown as optimizing the parameters of the LPF given a set of training data and set values for the mean and the covariance such that the performance mismatch of the LPF-based GMM and the original GMM is minimized. The mechanism for choosing the parameters of LPF is performed in a number of iterations. First, our approximation is restricted to a bounded interval \((0, c)\) for which \(f(z)\) is defined as

\[
f(z) = \begin{cases} 
\exp(-z), & \text{if } z < c \\
0, & \text{if } z \geq c
\end{cases} \quad (19)
\]

The value of \(c\) is obtained such that \(f(z)\) can achieve similar performance with that of the GMM classifier using the original unbounded exponential function and for our given training data-set. Fig. 3 shows the three possible LPF functions and their corresponding parameters \((a, b, c, k1, \text{and } k2)\). For \(f_1(z)\), \(a\) is obtained by performing a grid search in the range \((0, c)\) such that for our given training data-set, the performance mismatch between LPF-based GMM and the original GMM is minimized. For \(f_2(z)\), the parameter \(b\) is set to be equal to \(c (b = c)\). The value of \(a\) is again obtained by performing a grid search in the range \((0, b)\) such that for our given training data-set, the performance mismatch between LPF-based GMM and the original GMM is minimized. For \(f_3(z)\), the selection process for \(a\) and \(c\) is similar to that of \(a\) and \(b\) for the case of \(f_2(z)\). Once parameters \(a\) and \(c\) are obtained, \(b\) is evaluated by performing a grid search in the range \((a, c)\). Once \(b\) is found, parameters \(k1\) and \(k2\) can be easily extracted, as shown in Fig. 3 (right).

In order to evaluate the effect of using LPF-based mixture models on the classification performance, the classification results using the analog exponential function is compared with those using the three piecewise functions for the seven data-sets described in Table I. Hepatitis, Crabs, Breast, Iris, Odor 1, and 2 are the PCA projected data-sets with dimension of 5, 3, 3, 3, and 5, respectively, while for Vehicle data-set the original dimensionality \((d = 18)\) is kept unchanged. Ten-fold cross validation is again used to obtain the average performance of the GMM classifiers. The results of this comparison are illustrated in Table IV, while Table V shows an example of the optimized parameters of LPF for the seven data-sets.

From Table IV, we can conclude that the piecewise approximation can achieve comparable performance to the one obtained using an analog exponential function. The number of pieces required in the piecewise approximation depends upon the complexity of the classification problem at hand. For instance, \(f_1(z)\) approximation is sufficient for the Hepatitis data-set while \(f_2(z)\) approximation is required for Crabs, Breast, Iris, and Odor 1 data-sets and \(f_3(z)\) is required for the Vehicle and Odor 2 data-sets.

In order to simplify even further the GMM digital VLSI implementation, we approximated the different slope coefficients
of the piecewise functions \( k_1, k_2 \) using power-of-two coefficients \( (2^{-n}) \). This would enable for considerable hardware simplification of functions \( f_1(z) \) by only using right-shift operations instead of multiplications. The power-of-two slope coefficients are illustrated in Table V (number shown between brackets). The GMM performance was evaluated again using power-of-two coefficients and was found to be exactly the same as the one using real coefficients.

### C. What Is the Required Precision?

The previous sections have addressed many issues related to hardware simplification of GMM implementation. The linear piecewise approximation and the use of power-of-two coefficients will enable significant silicon area saving resulting in very compact arithmetic units making the prospect of building single VLSI chips solutions particularly promising for real-time decisions. However, implementing GMM with hardware of limited precision will affect the performance due to the quantization errors of continuous parameters of the classifier. Carefully studying the precision requirement for the application at hand is, therefore, very important [21]. GMM coefficients are then coded with the minimum required precision without affecting the performance of the classifier too much. In order to select appropriate precision, we have simulated the classification performance as a function of the number of bits in two-complement coding for all the data-sets previously reported. After training the classifier, the GMM coefficients for each classification problem including the parameters of linear piecewise function are normalized and uniformly quantized with \( b \) bits of precision by assigning \( N = 2^b \) uniform intervals over the coefficients range. The number of bits \( b \) is varied for each application and the classification performance are again evaluated. Fig. 6(a) shows the classification performance of the GMM classifier as a function of the number of bits for the seven data-sets described in Table I. It is obvious from Fig. 6(a) that, on one hand, the quantization error drastically decreases the classification performance. On the other hand, the number of bits required depends on the classification application at hand. In order to maintain acceptable performance, 8 bits are sufficient to achieve performances that are equivalent to analog coefficients for hepatitis breast and Iris data-sets. However, 10 bits are required for both Crabs and Odor 1 data-sets, while 12 bits are needed to achieve the same performance as analog coefficient for Odor 2 and Vehicle data-sets, respectively. It should be noted that using 10-bits for Odor 2 and Vehicle data-sets will not drastically affect the performance as only a decrease of less than 0.4% and 1% are observed. It must be, however, noted that the hardware complexity is increased by almost 25% when using 12-bit instead of 10-bit precision as illustrated by Fig. 6(b). This figure was accurately obtained by synthesizing the Verilog code of GMM architecture and reporting the results of the gate count for each number of bits. It is clear that increasing the number of bits from 10 to 12 bits will very slightly improve the performance for Odor 2 and Vehicle data-sets but will very drastically affect the gate count. In our hardware implementation, it was decided that 10 bits is a fairly reasonable precision for Odor 1 and Odor 2 applications (which are our main focus applications).

### IV. Digital VLSI Architecture

Fig. 7 shows the functional block diagram of the overall GMM classifier system. The architecture includes two main registers \( \text{Reg-X} \) and \( \text{Reg-GMM} \) used to store the input pattern \( x \) and the GMM parameters \( (\mu, G, K) \), respectively. The main computational block of the system is the GMM processor used to calculate \( \varphi(x|C_k)p(C_k) \) for each class in turn and, hence, makes the final class assignment decision for a given input pattern.

The GMM processor includes a serial parallel vector matrix multiplier, a square and multiplier units, a LPF unit, a winner-takes-all (WTA) circuit, and two accumulators. A 10-bit data bus is used in order to load the GMM parameters and the test vector data using the load signals shown on Fig. 7. This same 10-bit bus is used to load an internal control register used to configure the GMM processor for different modes of operation as will be described in upcoming sections. Once all the parameters are loaded, the GMM processor can operate on real-time basis by processing data in a systolic fashion. Initially, the \( G_j \)

### TABLE IV

<table>
<thead>
<tr>
<th>Data-sets</th>
<th>( \exp(z) )</th>
<th>( f_1(z) )</th>
<th>( f_2(z) )</th>
<th>( f_3(z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis</td>
<td>86.3</td>
<td>86.3</td>
<td>0%</td>
<td>86.3</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Crabs</td>
<td>100</td>
<td>99.5</td>
<td>0.5%</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Breast</td>
<td>93.2</td>
<td>90.5</td>
<td>2.8%</td>
<td>93.2</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Vehicle</td>
<td>64.1</td>
<td>51.5</td>
<td>12.6%</td>
<td>60.4</td>
</tr>
<tr>
<td></td>
<td>3.7%</td>
<td>3.7%</td>
<td>0%</td>
<td>64.1</td>
</tr>
<tr>
<td>Iris</td>
<td>98</td>
<td>91.3</td>
<td>6.7%</td>
<td>97.3</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Odor 1</td>
<td>100</td>
<td>97.5</td>
<td>2.5%</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Odor 2</td>
<td>94.1</td>
<td>77.3</td>
<td>17.8%</td>
<td>91.4</td>
</tr>
<tr>
<td></td>
<td>2.8%</td>
<td>2.8%</td>
<td>0%</td>
<td>92.3</td>
</tr>
</tbody>
</table>

### TABLE V

| Example of Parameters Value of Different Piecewise Functions. The Coefficients for \( f_2(z) \) Approximation Are Not Reported for Crabs, Breast, Iris and Odor 1 Data-Sets Because \( f_2(z) \) Approximation Is Sufficient While Both \( f_2(z) \) and \( f_3(z) \) Coefficients Are Not Reported for Hepatitis Data-Set Because \( f_2(z) \) Approximation Is Sufficient. NA Stands for Not Applicable. Numbers Shown Between Brackets Are the Power of Two Approximations Used for the Different Coefficients |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| \( f_1(z) \)    | \( f_2(z) \)    | \( f_3(z) \)    |                 |                 |                 |                 |
| Hepatitis \( a = 28 \) | \( a = 3, b = 7 \) | \( k_1 = 1.24 \times 10^{-2} \) | \( k_2 = 4.4 \times 10^{-4} \) | \( k_3 = 5.22 \times 10^{-7} \) | NA              | NA              |
| Crabs \( a = 5 \)    | \( k_1 = 4.6 \times 10^{-3} \) | \( k_2 = 1.01 \times 10^{-5} \) | \( k_3 = 2.44 \times 10^{-7} \) | \( k_4 = 5.22 \times 10^{-7} \) | NA              | NA              |
| Breast \( a = 5 \)  | \( k_1 = 1.26 \times 10^{-2} \) | \( k_2 = 2.00 \times 10^{-6} \) | \( k_3 = 3.5 \times 10^{-3} \) | \( k_4 = 4.6 \times 10^{-3} \) | NA              | NA              |
| Vehicle \( a = 7.4 \) | \( k_1 = 9.6 \) | \( k_2 = 1.5 \times 10^{-3} \) | \( k_3 = 4.0 \times 10^{-3} \) | \( k_4 = 4.9 \times 10^{-6} \) | NA              | NA              |
| Iris \( a = 11 \)  | \( k_1 = 19 \) | \( k_2 = 2.00 \times 10^{-6} \) | \( k_3 = 3.5 \times 10^{-3} \) | \( k_4 = 4.6 \times 10^{-3} \) | NA              | NA              |
| Odor 1 \( a = 11 \) | \( k_1 = 19 \) | \( k_2 = 2.00 \times 10^{-6} \) | \( k_3 = 3.5 \times 10^{-3} \) | \( k_4 = 4.6 \times 10^{-3} \) | NA              | NA              |
| Odor 2 \( a = 6 \)  | \( k_1 = 23 \) | \( k_2 = 1.5 \times 10^{-3} \) | \( k_3 = 4.0 \times 10^{-3} \) | \( k_4 = 4.9 \times 10^{-6} \) | NA              | NA              |
Fig. 6. (a) Performance (%) of GMM classifier using different number of bits for the five data-sets. (b) Gate count obtained for different number of bits. The reported results are obtained using the synthesis tools of Synopsys and repeated for each individual precision.

Fig. 7. Functional blocks of the GMM classifier system. The control unit is implemented using a finite-state machine. Information related to the GMM classifier such as the total number of Gaussian models and the number of Gaussian models for each class are stored in the two 20-bit registers within the control unit. The square calculation is performed using an array multiplier. The latency and throughput of the system are 0.79 μs and 0.63 μs, respectively, for an operating frequency of 80 MHz and for ten Gaussian models.

The vector-matrix multiplier is used to calculate $y_j$, where $y_j = \mathbf{x} - \mathbf{\mu}_j$. The vector-matrix multiplier is optimized for our specific need of vector-triangular matrix multiplication. Fig. 8 describes the vector-triangular matrix multiplication for $d = 3$.

A. Serial-Parallel Vector-Matrix Multiplier

The vector-matrix multiplier is used to calculate $y_j = s_j^T G_j$, where $s_j = \mathbf{x} - \mathbf{\mu}_j$. The vector-matrix multiplier is optimized for our specific need of vector-triangular matrix multiplication. Fig. 8 describes the vector-triangular matrix multiplication for $d = 3$.

The multiplication can be simplified by decomposing the $G$ matrix into row vectors which are then multiplied by $s_i$ components. Once this is achieved, the final resulting row vector $y$ is obtained by summing-up the intermediate results row wise (refer to Fig. 8). One can note that $y_0$ just requires one multiplication while $y_2$ and $y_3$ require two and three multiplications, respectively. Due to this property, we can first multiply $s_3$ with the $3^{rd}$ row of $G$ to generate $y_3$ and the partial results of $y_1$ and $y_2$ are temporarily accumulated. Next, $s_2$ is multiplied with the $2^{nd}$ row of $G$ and $y_2$ can be generated after accumulation. This multiplication was achieved using an efficient systolic architecture illustrated in Fig. 9, which permits to obtain a good tradeoff between operation speed and chip area. The elements of vector

$$\begin{bmatrix} s_1 & s_2 & s_3 \end{bmatrix} \times \begin{bmatrix} e_{11} & 0 & 0 \\ e_{21} & e_{22} & 0 \\ e_{31} & e_{32} & e_{33} \end{bmatrix}$$

WTA circuit. One should note that all these building blocks were thoroughly optimized before the final GMM processor was designed.
are fed into the multiplier serially while the row components of \( G \) are provided in parallel.

Fig. 9(a) describes the vector-matrix multiplier. For simplicity reasons, the figure only shows a maximum dimension of 5. Note that it is possible to use the structure for a lower dimension since zero components can be inserted in nonutilized blocks. Fig. 9(b) shows the computation sequences over five clock cycles. At the first clock cycle \( (t_1) \), \( g_{35} \) is fed to the vector matrix multiplier together with \( s_5 \). During this first cycle, output \( y_5 = s_5 g_{35} \) is obtained. At the next clock cycle \( (t_2) \), \( g_{45} \) is fed to the vector matrix multiplier together with \( s_4 \) and \( y_4 = s_4 g_{45} + s_{45} g_{54} \) is obtained. The procedure is continued until the resulting vector components \( y_t \) are obtained in five clock cycles.

**B. Linear Piecewise Function Unit**

In Section III, we have discussed a hardware-friendly GMM classifier using LPF instead of exponential function. Three different linear piecewise functions were proposed. Selecting appropriate approximation depends on the classification problem at hand. In our VLSI implementation, a single circuit can be configured to realize any of the three approximations.

The implementation of \( f_1(z) \) is straightforward. A comparator can be used to compare \( z \) and \( a \). If \( z < a \), the output is 1, otherwise, the output is 0. For \( f_2(z) \), two comparators are required. If \( z < a \) or \( z \geq b \), the output is 1 or 0, respectively. If \( a \leq z < b \), \( -k_3(z-b) \) will be calculated. In this case, (15) can be rewritten as:

\[
\varphi(\mathbf{x}|C_k)\psi(C_k) = \sum_{j=1}^{M} K_j k_3(b-z)
\]

However, the multiplication of \( k_3 \) is not necessary when calculating \( \varphi(\mathbf{x}|C_k)\psi(C_k) \) because the GMM classifier just compares \( \varphi(\mathbf{x}|C_k)\psi(C_k) \) in turn and selects the maximum component which will correspond to the selected class to be assigned to the input pattern. Since \( k_1 \) is the same for all components, the comparison result is not affected by removing \( k_1 \) from the equation and, hence, the classification result is not altered. As a consequence, \( f_2(z) \) can be rewritten as:

\[
f_2(z) = \begin{cases} 
1, & \text{if } z < a \\
1 - 2^{m-n}(b-z), & \text{if } a \leq z < b \\
0, & \text{if } z \geq b
\end{cases}
\]

In order to implement \( f_3(z) \), three comparators are required. When \( a \leq z < c \), \( -k_3(z-b) \), or \( -k_2(z-c) \) will be calculated. In this case, \( k_1 \) and \( k_2 \) cannot be omitted when calculating \( \varphi(\mathbf{x}|C_k)\psi(C_k) \). However, if the power-of-two approximation is used as introduced in Section III, we approximate \( k_1 \) by \( 2^{-m} \) and \( k_2 \) by \( 2^{-n} \) with \( (m < n) \). Therefore, \( k_1 \) can be expressed as \( k_1 = (2^{-m})k_2 \) and \( f_3(z) \) can be simplified to:

\[
f_3(z) = \begin{cases} 
1, & \text{if } z < a \\
1 - 2^{m-n}(b-z), & \text{if } a \leq z < b \\
(c-z), & \text{if } b \leq z < c \\
0, & \text{if } z \geq c.
\end{cases}
\]

It is obvious that calculating \( f_3(z) \) is now simplified to a set of comparison, subtraction, and shifting operations.

Fig. 10 shows the digital architecture of the LPF unit. \( R1-R6 \) are registers required to store the different parameters of the piecewise approximation. \( R7 \) is the input register which is loaded by the input data \( z \) (40 bits) when the \textit{load} signal is enabled. \( SR1 \) is a 40-bit shift register with set and reset options \((S,R)\). \( SR1 \) can shift the data by a number of bits set by the value stored in register \( R6 \). Three comparators \( C1-C3 \) are also required in this implementation, allowing to compare the input data \( z \) with the values stored in registers \( R1-R3 \), respectively. The output of the comparator is switched high if the input \( z \) is higher than the value stored in the corresponding register. The proposed circuit shown in Fig. 10 operates in three different modes enabling to realize all the three possible linear piecewise functions: \( f_1(z), f_2(z), \) and \( f_3(z) \).

- **Mode 1** (\( f_1(z) \)): \( a \) is stored in both \( R1 \) and \( R2 \). \( C1 \) compares input \( z \) with \( a \). If \( z < a \), the output of \( C1 \) is low and the output of \( SR1 \) will be set to 1. If \( z \geq a \), the output of \( C2 \) is high and this signal is used to reset \( SR1 \).
- **Mode 2** (\( f_2(z) \)): \( a \) is stored in \( R1 \) and \( R3 \) while \( b \) is stored in \( R2 \) and \( R5 \). \( C1 \) operates similarly to the case described in mode 1. If \( z \geq a \), \( C2 \) will compare \( z \) and \( b \). If \( z \geq b \)
SR1 will be reset, otherwise z will be loaded to register R0 and C3 will compare z with the content of R3. \((b - z)\) is then calculated and loaded into SR1 \((C3\) high). Finally, the output of SR1 is \((b - z)\) if \(a \leq z < b\).

- Mode 3 \((f_3(z))\): a, b, and \(b'\) are stored in R1, R3, and R4, respectively, while c is stored in R2 and R5. R6 stores the value of \((n - m)\). If \(z < a\), C1 will set SR1. The latter will be reset by C2 if \(z \geq b\). In the case where \(b \leq z < c\), the operation is the same as that of mode 2 and the output of SR1 will be \((c - z)\). If now \(a \leq z < b\), the output of C3 will be low and, thus, the subtractor will calculate \((b' - z)\) and the result is shifted to the left by \((n - m)\) positions. As a consequence, the output of SR1 is set to \(2^{n-m}(b' - z)\).

C. WTA Circuit

In the GMM classifier, the values \(\psi(x|C_j)\psi(C_j)\) need to be compared one-by-one before making the final classification decision. The maximum value is selected and its corresponding class is assigned to the test pattern. A WTA circuit is, therefore, required to perform this last step, as was illustrated in Fig. 7. Fig. 11 shows the architecture of the WTA circuit used for this purpose. Initially, R10 and R11 are reset and the initial control sequence stored in the shift register SR2 is “10000” as illustrated in Fig. 11. The values in the shift register are shifted at each rising edge of the enable signal which is done each time a new value \(\psi(x|C_j)\psi(C_j)\) is available. Each bit within the 5-bit control sequence stored in SR2 is responsible for enabling one D-FF. By shifting the control sequence in SR2 the output of the comparator is loaded in a new D-FF. The WTA circuit operation starts by comparing the first component \(\psi(x|C_1)\psi(C_1)\) to “0.” The comparison result is stored in D1. Since \(\psi(x|C_1)\psi(C_1)\geq 0\), the output of the comparator is high which will enable register R11 using the feedback connection and, hence, allowing to store, in the next cycle, \(\psi(x|C_1)\psi(C_1)\) in register R11 and \(\psi(x|C_1)\psi(C_1)\) in register R10. \(\psi(x|C_1)\psi(C_1)\) is then compared to \(\psi(x|C_2)\psi(C_2)\) and D2 will store this new comparison result. If the result is high, the same procedure is repeated as described previously and D2 is loaded with a value of “1” which will consequently reset the LSB bit (D1) using the output of the OR gate as shown in Fig. 11. If \(\psi(x|C_1)\psi(C_1) > \psi(x|C_2)\psi(C_2)\) the output of the comparator will be low and R11 will keep \(\psi(x|C_1)\psi(C_1)\), which corresponds to the largest value and D2 is loaded with a value of “0.” The process is repeated at each active enable signal and at the end all flip-flops D1-D5 will be zeroed except for one flip-flop which will correspond to \(\psi(x|C_j)\psi(C_j)\). The nonzeroed flip-flop will indicate the index of the selected class for the processed input pattern. The whole GMM processor is pipelined, while the comparison of all \(\psi(x|C_k)\psi(C_k)\) is performed iteratively and the final classification result is stored in a 5-bit register of the WTA circuit.

D. Reconfigurability

The proposed GMM classifier architecture is expected to operate for different classification problems. However, the dimension and the total number of Gaussian models varies from one classification problem to another. It was shown in the previous section that the vector-matrix multiplier can be used for all dimensions by adding zero components in the case where a lower dimensionality is required. A programmable control unit was designed in order to deal with the requirement of a different number of Gaussian models. This is realized using the control unit shown in Fig. 7, which is a FSM. The latter contains \(2 \times 20\)-bit registers used to store the control parameters responsible for reconfiguring the GMM processor. One register stores the value of the total number of Gaussian models, and the second register stores the value of the number of Gaussian models for each class. The size of the register array is designed to store the parameters of a maximum of ten Gaussian models. The output of the WTA is 5 bits (1 bit for each class). Thus, the GMM processor can operate on variable numbers of Gaussian models (2 to 10), variable number of classes (2 to 5) and variable dimensions (1 to 5).

E. System Operation and Experimental Results

The system operation can be divided into three phases: 1) loading the parameters of the classifier; 2) loading the input patterns \(x\) to be processed; and 3) performing the classification. When the signal Load-GMM is high, the parameters of the classifier will be loaded into Reg-GMM serially through a 10-bit bus (Fig. 7). Next, Load-GMM is set low and Load-x is set high enabling to load the input patterns into Reg-x. The system is then enabled and the classification process starts. The parameters of the classifier are loaded only once and input patterns are sequentially processed without the need to load the parameters again. The timing diagram corresponding to the different phases of operation is illustrated in Fig. 12. Because there is a total of 235 10-bit parameters (including parameters of the GMM classifier, exponential unit and parameters used by the control unit), 235 clock cycles are needed for the first phase. Loading the input pattern will only require five cycles for a maximum of five components (dimension of \(x\) is 5). Each component is loaded sequentially using the same 10-bit bus. Once the GMM parameters and the
input patterns are loaded, the classification process is initiated. The classification process can be divided into a number of basic operations namely: subtraction, vector matrix multiplication, square accumulation, LPF calculation, multiplication accumulation, and finally, WTA calculation. In order to increase the speed of the GMM classifier, a pipelining strategy is adopted as shown in Fig. 12. Processing a new PGM \((y(j|x))\) starts every five clock cycles and each basic operation requires five clock cycles to perform the corresponding calculation. All of the basic operations described previously are performed in parallel with at most one clock cycle delay. For Gaussian mixtures requiring \(j\) components, \(5j\) clock cycles are required before the final \(\phi(x|C_k)\phi(C_k)\) result is available. At this stage, the WTA is enabled and will process the results sequentially every multiple of five clock cycles depending on the number of parameters in the mixture. The total number of clock cycles required in this classification process depends on the total number of Gaussian models. For \(M\) models, the classification is performed in \(5M + 13\) clock cycles. Since processing is performed in a pipeline, the loading of a new pattern starts \(5M - 5\) cycles while processing a new pattern starts after \(5M\) cycles as illustrated in Fig. 12.

The GMM classifier was implemented using a 0.25-\(\mu\)m CMOS process. The gate level circuit was synthesized using Synopsys. The final circuit was designed using Encounter tools for automatic placement and routing. The chip implements the recall operation of the GMM classifier as described in Section II. It includes GMM coefficient storage memory together with local registers storing the GMM topology, FSM for reconfiguration and control of the GMM core classifier, and finally, the arithmetic unit implementing the previously described operations. Fig. 13 shows a microphotograph of the chip. The GMM processor occupies an area of about 1.69 mm\(^2\).

Before implementing the recall operation of any classification problem, the GMM core processor is first configured in terms of number of classes, number of Gaussian components per class, and the required linear piecewise approximation for the exponential calculation. A setup of a given topology requires loading the \(2 \times 20\) bit registers. Each time the topology needs to be changed, the content of the two registers needs to be loaded into the chip accordingly. In order to verify the correct operation of the chip, Verilog was used in order to generate test vectors from a high-level description of the design. The generated test vectors were then inserted within the test program of the digital tester. This has allowed us not only to verify the functionality of the design for different GMM topologies, but also to extract its performance for a real classification problem. Selective gas detection application reported as Odor 2 data-set was used as a test-bed for the chip operating as a GMM classifier. The tested classification application is related to the detection of five gases (CO, CH\(_4\), H\(_2\), mixture of CO and CH\(_4\), and mixture of CO and H\(_2\)) by using an array of in-house developed tin-oxide gas sensors. The classification performance are reported for 100 sample inputs and using the 3 possible piecewise linear approximation functions reported in Section III. Table VI reports the performance of the chip for the specified application and for different configurations. The latency and throughput of the system are dependant upon the total number of Gaussians used in the classification system. The chip operates at a clock frequency of 80 MHz. For odor 2 data-set, all the GMM parameters are loaded in less than 3 \(\mu\)s and the first pattern is classified in less than 0.8\(\mu\)s. The tested processing time corresponding to the classification of 100 patterns is less than 57 \(\mu\)s.
Successful operation of the architecture is demonstrated through simulation results as well as experimental tests in Section III. The parameters for the architecture have greatly helped in improving this paper.

TABLE VI

<table>
<thead>
<tr>
<th></th>
<th>GMM processor</th>
<th>Clock Frequency</th>
<th>GMM parameters load time</th>
<th>Latency</th>
<th>Test patterns</th>
<th>Classification accuracy</th>
<th>Processing time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.69mm²</td>
<td>80MHz</td>
<td>2.92μs</td>
<td>0.72μs</td>
<td>20CO, 20H₂, 20CH₄</td>
<td>f₁(z) 77.0%</td>
<td>56.25μs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20CO – CH₄, 20CO – H₂</td>
<td>f₂(z) 91.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>f₃(z) 92%</td>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUSION

In this paper, we presented a pattern recognition system based on a GMM classifier. Simulation results suggest that the GMM classifier presents the best classification performance with acceptable complexity when compared to KNN, MLP, RBF, and PPCA. An efficient digital VLSI implementation of the GMM classifier is proposed offering a very good balance between hardware complexity and operating speed using novel pipelining strategy as wel as hardware-friendly VLSI architecture. This hardware-friendly architecture is based on the use of power-of-two coefficients as well as LPF-based GMM. The proposed procedure consists of building an original GMM based on the training data-set and then optimizing the parameters of the LPF-based GMM using the classification performance as an optimization criteria. It was shown that our proposed technique permits to achieve good classification performance for the problems at hand while keeping the complexity of the overall system to an acceptable level. It is believed that further improvement can be achieved by studying the tradeoff involved in using diagonal covariance matrix and developing new learning algorithms for the desired probability function.

A prototype chip was designed using automatic placement and routing based on 0.25-μm technology occupying an area of 1.69 mm². Successful operation of the architecture is demonstrated through simulation results as well as experimental tests for gas identification application requiring ten Gaussian models and five classes. A classification performance of 92% was achieved for 100 input patterns processed in less than 57 μs. To the best of our knowledge, this prototype represents the first reported hardware implementation of a GMM classifier.

ACKNOWLEDGMENT

The authors would like to thank Dr. S. B. Belhouari for his help and technical discussion. They would also like to thank the anonymous reviewers for their very constructive review which has greatly helped in improving this paper.

REFERENCES


Minghua Shi received the B.S. degree in electronic engineering from East China University of Science and Technology, Shanghai, China, in 2002. He is currently pursuing the Ph.D. degree under the supervision of Prof. A. Bermak from Electronic and Computer Engineering Department at the Hong Kong University of Science and Technology (HKUST), Clear Water Bay, Kowloon, Hong Kong. His research interests include hardware implementation of pattern recognition algorithms for electronic nose applications.
Amine Bermak (M’99–SM’04) received the M.Eng. and Ph.D. degrees in electronic engineering from Paul Sabatier University, Toulouse, France, in 1994 and 1998, respectively.

While completing his doctoral work, he was part of the Microsystems and Microstructures Research Group, French National Research Center LAAS-CNRS, Toulouse, France, where he developed a 3-D VLSI chip for artificial neural network classification and detection applications. He joined the Advanced Computer Architecture research group, York University, York, U. K., where he was a Postdoctoral Researcher working on VLSI implementation of CMM neural network for vision applications in a project funded by British Aerospace. In November 1998, he joined Edith Cowan University, Perth, Australia, first as a Research Fellow working on smart vision sensors, then as a Lecturer and a Senior Lecturer with the School of Engineering and Mathematics. He is currently an Assistant Professor with the School of Science and Technology (HKUST), Clear Water Bay, Kowloon, Hong Kong, where he is also serving as the Associate Director of the Computer Engineering Programme. His research interests include VLSI circuits and systems for signal, image processing, sensors, and microsystems applications. He has published extensively on the previous topics in various journals, book chapters, and refereed international conferences.

Dr. Bermak was the recipient of the Bechtel Foundation Engineering Teaching Excellence Award and the IEEE Chester Sall Award in 2004.