

# Kerneltron: Support Vector “Machine” in Silicon

Roman Genov, *Member, IEEE*, and Gert Cauwenberghs, *Senior Member, IEEE*

**Abstract**—Detection of complex objects in streaming video poses two fundamental challenges: training from sparse data with proper generalization across variations in the object class and the environment; and the computational power required of the trained classifier running real-time. The Kerneltron supports the generalization performance of a support vector machine (SVM) and offers the bandwidth and efficiency of a massively parallel architecture. The mixed-signal very large-scale integration (VLSI) processor is dedicated to the most intensive of SVM operations: evaluating a kernel over large numbers of vectors in high dimensions. At the core of the Kerneltron is an internally analog, fine-grain computational array performing externally digital inner-products between an incoming vector and each of the stored support vectors. The three-transistor unit cell in the array combines single-bit dynamic storage, binary multiplication, and zero-latency analog accumulation. Precise digital outputs are obtained through oversampled quantization of the analog array outputs combined with bit-serial unary encoding of the digital inputs. The 256 input, 128 vector Kerneltron measures 3 mm × 3 mm in 0.5 μm CMOS, delivers 6.5 GMACS throughput at 5.9 mW power, and attains 8-bit output resolution.

**Index Terms**—Analog array processors, analog-to-digital conversion (ADC), charge-injection device (CID), dynamic random-access memory (DRAM), matrix-vector multiplication (MVM), oversampling quantization, pattern recognition, support vector machines (SVMs), vector quantization (VQ).

## I. INTRODUCTION

SUPPORT VECTOR machines (SVMs) [1] offer a principled approach to machine learning combining many of the advantages of artificial intelligence and neural-network approaches. Underlying the success of SVMs are mathematical foundations of statistical learning theory [2]. Rather than minimizing training error (*empirical risk*), SVMs minimize *structural risk* which expresses an upper bound on the generalization error, i.e., the probability of erroneous classification on yet-to-be-seen examples. This makes SVMs especially suited for adaptive object detection and identification with sparse training data.

Real-time detection and identification of visual objects in video from examples is generally considered a hard problem for two reasons. One is the large degree of variability in the object class, i.e., orientation and illumination of the object or occlu-

sions and background clutter in the surrounding, which usually necessitates a large number of training examples to generalize properly. The other is the excessive amount of computation incurred during training, and even in run-time.

Support vector machines have been applied to visual object detection, with demonstrated success in face and pedestrian detection tasks [3]–[6]. Unlike approaches to object detection that rely heavily on hand-crafted models and motion information, SVM-based systems learn the model of the object of interest from examples and work reliably in absence of motion cues. To reduce the computational burden of real-time implementation to a level that can be accommodated with available hardware, a reduced set of features are selected from the data which also result in a reduced number of support vectors [5]. The reduction in implementation necessarily comes at a loss in classification performance, a loss which is more severe for tasks of greater complexity.

The run-time computational load is dominated by evaluation of a kernel between the incoming vector and each of the support vectors. For a large class of permissible kernels, which include polynomial splines and radial kernels, this computation entails matrix-vector multiplication in large dimensions. For the pedestrian detection task in unconstrained environments [5], highest detection at lowest false alarm is achieved for very large numbers (thousands) of input dimensions and support vectors, incurring millions of matrix multiply-accumulates (MAC) for each classification. The computation recurs at different positions and scales across each video frame.

The Kerneltron offers a factor 100–10 000 improvement in computational efficiency (throughput per unit power) over the most advanced digital signal processors available today. It affords this level of efficiency at the expense of specificity: the very large-scale integration (VLSI) architecture is dedicated to massively parallel kernel computation [7]. Speed can be traded for power dissipation. Lower power is attractive in portable applications of kernel-based pattern recognition, such as visual aids for the blind [8].

Section II briefly summarizes feature extraction and SVM classification for object detection in streaming video. Section III describes the architecture and circuit implementation of the Kerneltron. Experimental results, scalability issues, training, and application examples are discussed in Section IV.

## II. OBJECT DETECTION WITH SUPPORT VECTOR MACHINES

A support vector machine is trained with a data set of labeled examples. For pattern classification in images, relevant features are typically extracted from the training set examples using redundant spatial filtering techniques, such as overcomplete wavelet decomposition [4]. The classifier is trained on these feature vectors. In run time, images representing frames

Manuscript received September 15, 2002. This work was supported by NSF IIS-0209289 (Universal Access), ONR N00014-99-1-0612, ONR/DARPA N00014-00-C-0315 and WatchVision Corporation. The chip was fabricated through the MOSIS service.

R. Genov was with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 USA. He is now with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: roman@eecg.toronto.edu).

G. Cauwenberghs is with the Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218 USA (e-mail: gert@bach.ece.jhu.edu).

Digital Object Identifier 10.1109/TNN.2003.816345

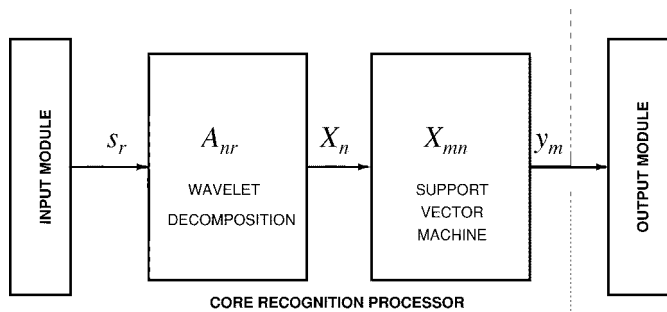


Fig. 1. Functional block diagram of the SVM classifier. The core of the system is a support vector machine processor for general object detection and classification. An overcomplete wavelet decomposition of the incoming sensory data at the input generates redundant input features to the SVM, providing for robust and relatively invariant classification performance.

of streaming video are scanned by moving windows of different dimensions. For every unit shift of a moving window, a wavelet feature vector is computed and presented to the SVM classifier to produce a decision. The general block diagram of such a system is outlined in Fig. 1. A brief functional description of the major components follows next.

#### A. Overcomplete Wavelet Decomposition

An overcomplete wavelet basis enables the system to handle complex shapes and achieve a precise description of the object class at adequate spatial resolution for detection [4]. The transformation of the sensory data  $s$  into the feature vector  $\mathbf{X}$  is of the linear form

$$X_n = \sum_{r=1}^R A_{nr} s_r, \quad n = 1, \dots, N \quad (1)$$

where the wavelet coefficients  $A_{nr}$  form an overcomplete basis, i.e.,  $N > R$ .

In visual object detection overcomplete Haar wavelets have been successfully used on pedestrian and face detection tasks [4], [5]. Haar wavelets are attractive because they are robust and particularly simple to compute, with coefficients  $A_{nr}$  that are either  $-1$  or  $1$ .

#### B. Support Vector Classification

Classification of the wavelet transformed features is performed by an SVM [1]. From a machine learning theoretical perspective [2], the appealing characteristics of SVMs are as follows.

- 1) The learning technique generalizes well even with relatively few data points in the training set, and bounds on the generalization error can be directly estimated from the training data.
- 2) The only parameter that needs tuning is a penalty term for misclassification which acts as a regularizer [9] and determines a tradeoff between resolution and generalization performance [10].
- 3) The algorithm finds, under general conditions, a unique separating decision surface that maximizes the margin of the classified training data for best out-of-sample performance.

SVMs express the classification or regression output in terms of a linear combination of examples in the training data, in which only a fraction of the data points, called “support vectors,” have nonzero coefficients. The support vectors thus capture all the relevant data contained in the training set. In its basic form, a SVM classifies a pattern vector  $\mathbf{X}$  into class  $y \in \{-1, +1\}$  based on the support vectors  $\mathbf{X}^m$  and corresponding classes  $y_m$  as

$$y = \text{sign} \left( \sum_{m=1}^M \alpha_m y_m K(\mathbf{X}^m, \mathbf{X}) - b \right) \quad (2)$$

where  $K(\cdot, \cdot)$  is a symmetric positive-definite kernel function which can be freely chosen subject to fairly mild constraints [1]. The parameters  $\alpha_m$  and  $b$  are determined by a linearly constrained quadratic programming (QP) problem [2], [11], which can be efficiently implemented by means of a sequence of smaller scale, subproblem optimizations [3], or an incremental scheme that adjusts the solution one training point at a time [12]. Most of the training data  $\mathbf{X}^m$  have zero coefficients  $\alpha_m$ ; the nonzero coefficients returned by the constrained QP optimization define the support vector set. In what follows we assume that the set of support vectors and coefficients  $\alpha_m$  are given, and we concentrate on efficient run-time implementation of the classifier.

Several widely used classifier architectures reduce to special valid forms of kernels  $K(\cdot, \cdot)$ , like polynomial classifiers, multilayer perceptrons,<sup>1</sup> and radial basis functions [14]. The following forms are frequently used:

- 1) inner-product based kernels (e.g., polynomial; sigmoidal connectionist):

$$K(\mathbf{X}^m, \mathbf{X}) = f(\mathbf{X}^m \cdot \mathbf{X}) = f \left( \sum_{n=1}^N X_{mn} X_n \right) \quad (3)$$

- 2) radial basis functions ( $L_2$  norm distance based)

$$K(\mathbf{X}^m, \mathbf{X}) = f(\|\mathbf{X}^m - \mathbf{X}\|) = f \left( \left( \sum_{n=1}^N |X_{mn} - X_n|^2 \right)^{\frac{1}{2}} \right) \quad (4)$$

where  $f(\cdot)$  is a monotonically nondecreasing scalar function subject to the Mercer condition on  $K(\cdot, \cdot)$  [2], [9].

With no loss of generality, we concentrate on kernels of the inner product type (3), and devise an efficient scheme of computing a large number of high-dimensional inner-products in parallel. Computationally, the inner-products comprise the most intensive part in evaluating kernels of both types (3) and (4). Indeed, radial basis functions (4) can be expressed in inner-product form

$$f(\|\mathbf{X}^m - \mathbf{X}\|) = f \left( (-2\mathbf{X}^m \cdot \mathbf{X} + \|\mathbf{X}^m\|^2 + \|\mathbf{X}\|^2)^{\frac{1}{2}} \right) \quad (5)$$

<sup>1</sup>With logistic sigmoidal activation function, for particular values of the threshold parameter only.

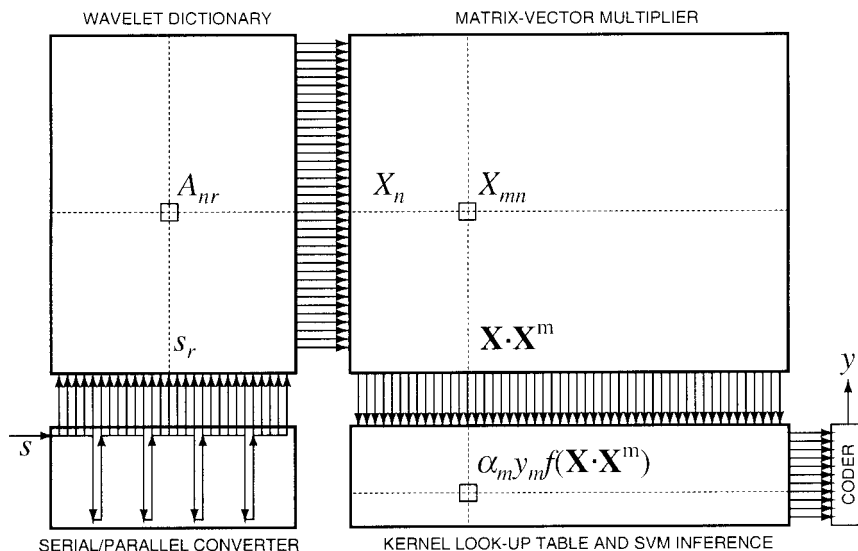


Fig. 2. Architecture of the core recognition processor, combining overcomplete wavelet decomposition with generalized support vector machine classification. Communication with outside modules is through a serial digital input/output interface for maximal flexibility and programmability, while the core internal computations are parallel and analog for optimal efficiency.

where the last two terms depend only on either the input vector or the support vector. These common terms are of much lower complexity than the inner-products, and can be easily precomputed or stored in peripheral registers.

The computation of the inner-products takes the form of matrix-vector multiplication (MVM),  $\sum_{n=1}^N X_{mn} X_n$ ;  $m = 1, \dots, M$ , where  $M$  is the number of support vectors. For large scale problems as the ones of interest here, the dimensions of the matrix  $M \times N$  are excessive for real-time implementation even on a high-end processor. As a point of reference, consider the pedestrian and face detection task in [5], for which the feature vector length  $N$  is 1326 wavelets per instance, and the number of support vectors  $M$  is in excess of 4000. To cover the visual field over the entire scanned image at reasonable resolution (500 image window instances through a variable resolution search method) at video rate (30 frames/s), a computational throughput of  $75 \times 10^9$  multiply-and-accumulate operations/s, is needed. The computational requirement can be relaxed through simplifying and further optimizing the SVM architecture for real-time operation, but at the expense of classification performance [4], [5].

### III. KERNELTRON: MASSIVELY PARALLEL VLSI KERNEL MACHINE

#### A. Core Recognition VLSI Processor

At the core of the system is a recognition engine, which efficiently implements kernel-based algorithms, such as SVMs, for general pattern detection and classification. The implementation focuses on inner-product computation in a parallel architecture.

Both wavelet and SVM computations are most efficiently implemented on the same chip, in a scalable VLSI architecture as illustrated schematically in Fig. 2. The diagram is the floorplan

of the Kerneltron, with matrices projected as two-dimensional (2-D) arrays of cells, and input and output vector components crossing in perpendicular directions alternating from one stage to the next. This style of scalable architecture also supports the integration of learning functions, through local outer product parameter updates [13], compatible with the recently developed incremental SVM learning rule [12]. The architecture maintains low input-output data rate. Digital inputs are fed into the processor through a properly sized serial/parallel converter shift register. A unit shift of a scanning moving window in an image corresponds to one shift of a new pixel per classification cycle, while a single scalar decision is produced at the output.

The classification decision is obtained in digital domain by thresholding the weighted sum of kernels. The kernels are obtained by mapping the inner-products  $\mathbf{X} \cdot \mathbf{X}^m$  through the function  $f(\cdot)$  stored in a lookup table.

By virtue of the inner-product form of the kernel, the computation can be much simplified without affecting the result. Since both the wavelet feature extraction and the inner-product computation represent linear transformations, they can be collapsed into a single linear transformation by multiplying the two matrices

$$W_{mr} = \sum_{n=1}^N X_{mn} A_{nr}. \quad (6)$$

Therefore, the architecture can be simplified to one that omits the (explicit) wavelet transformation, and instead transforms the support vectors.<sup>2</sup> For simplicity of the argument, we proceed with the inner-product architecture excluding the overcomplete wavelet feature extraction stage, bearing in mind that the approach extends to include wavelet extraction by merging the two matrices.

<sup>2</sup>Referred to the input prior to wavelet transformation, support vectors  $\mathbf{s}_m$  need to be transformed twice:  $W_{mr} = \sum_{n=1}^N \sum_{p=1}^S A_{np} A_{nr} s_{mp}$ .

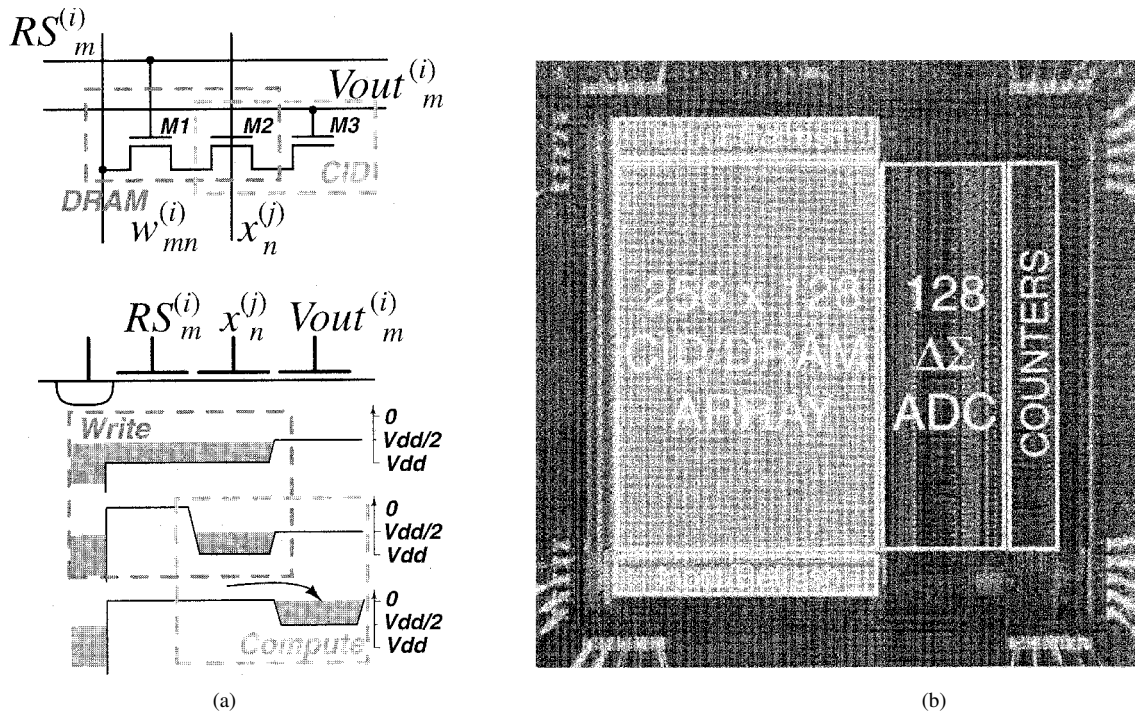


Fig. 3. (a) CID computational cell with integrated DRAM storage. Circuit diagram, and charge transfer diagram for active write and compute operations. (b) Micrograph of the *Kerneltron* prototype, containing containing an array of  $256 \times 128$  CID/DRAM cells, and a row-parallel bank of 128 algorithmic  $\Delta\Sigma$  ADCs. Die size is  $3 \text{ mm} \times 3 \text{ mm}$  in  $0.5 \mu\text{m}$  CMOS technology.

### B. Internally Analog, Externally Digital Computation

Computing inner-products between an input vector  $\mathbf{X}$  and template vectors  $\mathbf{W}_m$  in parallel is equivalent to the operation of matrix-vector multiplication (MVM)

$$Y_m = \sum_{n=0}^{N-1} W_{mn} X_n \quad (7)$$

with  $N$ -dimensional input vector  $\mathbf{X}$ ,  $M$ -dimensional output vector  $\mathbf{Y}$ , and  $M \times N$  matrix of coefficients  $W_{mn}$ . The matrix elements  $W_{mn}$  denote the support vectors  $X_{mn}$ , or the wavelet transformed support vectors (6) for convenience of notation.<sup>3</sup>

The approach combines the computational efficiency of analog array processing with the precision of digital processing and the convenience of a programmable and reconfigurable digital interface. The digital representation is embedded in the analog array architecture, with matrix elements stored locally in bit-parallel form

$$W_{mn} = \sum_{i=0}^{I-1} 2^{-i-1} w_{mn}^{(i)} \quad (8)$$

and inputs presented in bit-serial fashion

$$X_n = \sum_{j=0}^{J-1} \gamma_j x_n^{(j)} \quad (9)$$

where the coefficients  $\gamma_j$  are assumed in radix two, depending on the form of input encoding used. The MVM task (7) then decomposes into

$$Y_m = \sum_{n=0}^{N-1} W_{mn} X_n = \sum_{i=0}^{I-1} 2^{-i-1} Y_m^{(i)} \quad (10)$$

with MVM partials

$$Y_m^{(i)} = \sum_{j=0}^{J-1} \gamma_j Y_m^{(i,j)} \quad (11)$$

and

$$Y_m^{(i,j)} = \sum_{n=0}^{N-1} w_{mn}^{(i)} x_n^{(j)}. \quad (12)$$

The binary-binary partial products (12) are conveniently computed and accumulated, with zero latency, using an analog MVM array [15]–[18]. For this purpose, we developed a 1-bit multiply-and-accumulate CID/DRAM cell.

### C. CID/DRAM Cell and Array

The unit cell in the analog array combines a charge injection device (CID) [19] computational element [17], [18] with a DRAM storage element. The cell stores one bit of a matrix element  $w_{mn}^{(i)}$ , performs a one-quadrant binary-unary (or binary-binary) multiplication of  $w_{mn}^{(i)}$  and  $x_n^{(j)}$  in (12), and accumulates the result across cells with common  $m$  and  $i$  indexes. The circuit diagram and operation of the cell are presented in Fig. 3(a). An active charge transfer from M2 to M3 can only occur if there is nonzero charge stored, and if the potential on the gate of M2 drops below that of M3 [17]. The cell performs nondestructive computation since the transferred charge is sensed capacitively

<sup>3</sup>In the wavelet transformed case,  $s$  should be substituted for  $\mathbf{X}$  in what follows.

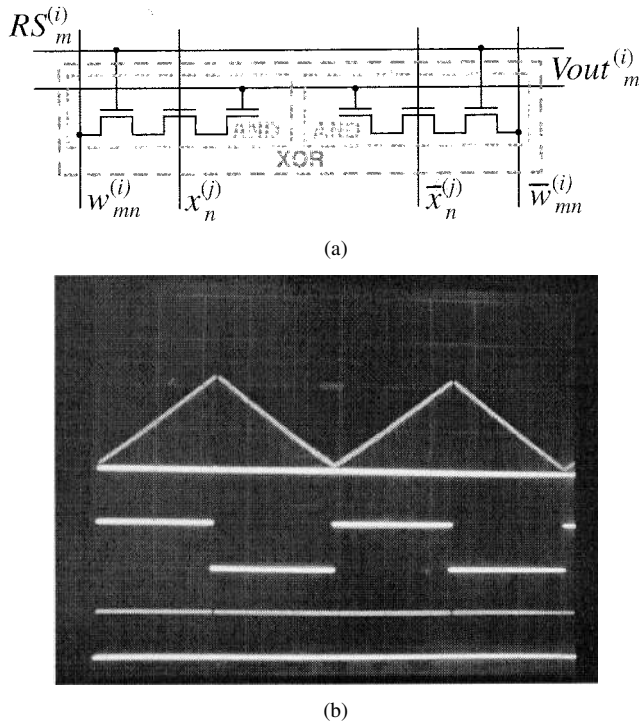


Fig. 4. (a) Two charge-mode AND cells configured as an exclusive-OR (XOR) multiply-and-accumulate gate. (b) Measured linearity of the computational array configured for signed multiplication on each cell (XOR configuration). Waveforms shown are, *top to bottom*: the analog voltage output,  $V_{out_m}^{(i)}$ , on the sense line (2 V dynamic range); input data (in common for both input,  $x_n^{(j)}$ , and weight,  $w_{mn}^{(i)}$ , shift register); and input shift register clock.

at the output. Thus, an array of cells performs (unsigned) binary-unary multiplication (12) of a matrix with elements  $w_{mn}^{(i)}$  and a vector with elements  $x_n^{(j)}$  yielding  $Y_m^{(i,j)}$ , for values of  $i$  in parallel across the array, and values of  $j$  in sequence over time. A  $256 \times 128$  array prototype using CID/DRAM cells is shown in Fig. 3(b).

To improve linearity and to reduce sensitivity to clock feedthrough, we use differential encoding of input and stored bits in the CID/DRAM architecture using twice the number of columns and unit cells as shown in Fig. 4(a). This amounts to exclusive-OR (XOR), rather than AND, multiplication on the analog array, using signed, rather than unsigned, binary values for inputs and weights,  $x_n^{(j)} = \pm 1$  and  $w_{mn}^{(i)} = \pm 1$ .

In principle, the MVM partials (12) can be quantized by a bank of flash analog-to-digital converters (ADCs), and the results accumulated in the digital domain according to (11) and (10) to yield a digital output resolution exceeding the analog precision of the array and the quantizers [20]. Alternatively, an oversampling ADC accumulates the sum (11) in the analog domain, with inputs encoded in unary format ( $\gamma_i = 1$ ). This avoids the need for high-resolution flash ADCs, which are replaced with single-bit quantizers in the delta-sigma loop.

#### D. Oversampling Mixed-Signal Array Processing

The precision of computation is limited by the resolution of the analog-to-digital converters (ADCs) digitizing the analog array outputs. The conventional delta-sigma ( $\Delta\Sigma$ ) ADC design paradigm allows to reduce requirements on precision of analog

circuits to attain high resolution of conversion, at the expense of bandwidth. In the presented architecture a high conversion rate is maintained by combining delta-sigma ADC with over-sampled encoding of the digital inputs, where the delta-sigma modulator integrates the partial multiply-and-accumulate outputs (12) from the analog array according to (11).

Fig. 5 depicts one row of matrix elements  $W_{mn}$  in the  $\Delta\Sigma$  oversampling architecture, encoded in  $I = 4$  bit-parallel rows of CID/DRAM cells. One bit of a unary-coded input vector is presented each clock cycle, taking  $J$  clock cycles to complete a full computational cycle (7). The data flow is illustrated for a digital input series  $x_n^{(j)}$  of  $J = 16$  unary bits.

Over  $J$  clock cycles, the oversampling ADC integrates the partial products (12), producing a decimated output

$$Q_m^{(i)} \approx \sum_{j=0}^{J-1} \gamma_j Y_m^{(i,j)} \quad (13)$$

where  $\gamma_j = 1$  for unary coding of inputs. Decimation for a first-order delta-sigma modulator is achieved using a binary counter.

#### E. Row-Parallel $\Delta\Sigma$ Algorithmic ADC

Higher precision can be obtained in the same number of cycles  $J$  by using a higher order delta-sigma modulator topology. However this drastically increases the implementation complexity. Instead, we use a modified topology shown in Fig. 6 that resamples the residue of the integrator after initial conversion. A sample-and-hold resamples the residue voltage of the integrator and presents it to the modulator input for continued conversion at a finer scale. The principle is analogous to extended counting [21] but avoids additional hardware by reusing the same  $\Delta\Sigma$  modulator to quantize the residue.

Similar to residue resampling in an algorithmic (or cyclic) ADC, for each resampling the scale of conversion subranges to the LSB level of the previous conversion. For a first-order incremental  $\Delta\Sigma$  ADC [22], resampling of the residue scales the range by a factor  $L$ , where  $L$  is the number of modulation cycles. If  $L$  is of radix two, i.e.,  $L = 2^\ell$ , then the subranging is conveniently accomplished in the architecture of Fig. 6 by shifting the bits in the decimating counter by  $\ell$  positions for every resampling of the residue.

Every resampling improves the output resolution by a factor  $L$ , or  $\ell$  bits, limited by noise and mismatch in the implementation. The effect of capacitance mismatch is minimized by using a ratio-insensitive scheme for resampling the residue [23]. The presented scheme is equivalent to algorithmic ADC, but avoids interstage gain errors without the need for precisely ratioed analog components.

The resampling of the residue in the oversampled ADC can be combined with correspondingly rescaling the coefficients  $\gamma_j$  in the input encoding. In principle, higher resolution digital inputs can be presented by unary encoding bits in groups of  $\ell$ , each covering  $L$  modulation cycles of the subranging oversampled ADC [23]. In Fig. 5, only the first four bits are unary encoded and presented in the first algorithmic cycle, with  $L = 16$ . With a single resampling of the residue, the  $\Delta\Sigma$  modulator obtains  $2\ell = 8$  bit effective resolution in  $2L = 32$  cycles.

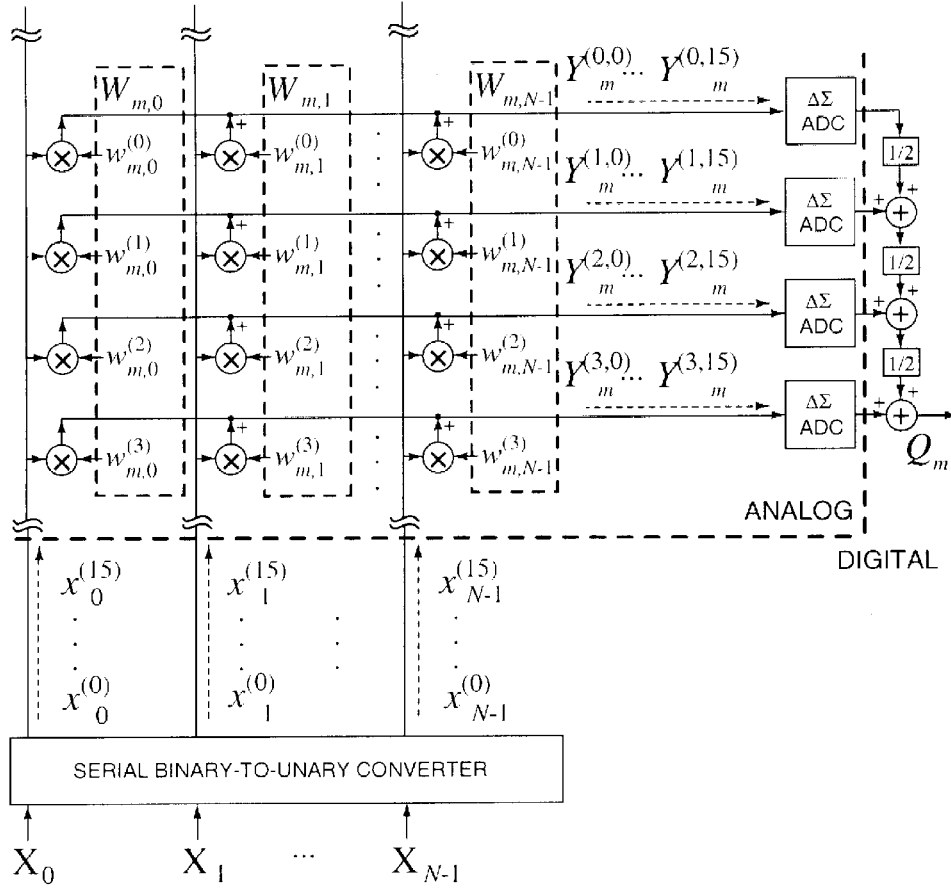


Fig. 5. Block diagram of one row of the matrix with binary encoded elements  $w_{mn}^{(i)}$ , for a single  $m$  and  $I = 4$ . Data flow of bit-serial unary encoded inputs  $x_n^{(j)}$  and corresponding partial product outputs  $Y_m^{(i,j)}$ , with  $J = 16$  bits. The full product for a single row  $Y_m^{(i)}$  is accumulated and quantized by a delta-sigma ADC. The final product is constructed in the digital domain according to (10).

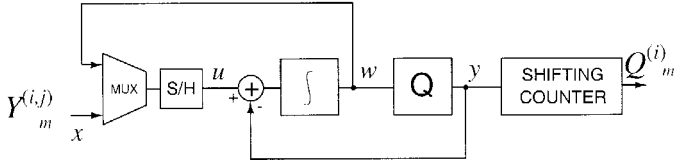


Fig. 6. Block diagram of  $\Delta\Sigma$  algorithmic ADC with residue resampling, including decimator. One ADC is provided for each row in the array architecture.

Additional gains in precision can be obtained by exploiting binomial statistics of binary terms in the analog summation (12) and [24]. In the present scheme, this would entail stochastic encoding of the digital inputs prior to unary oversampled encoding.

#### IV. EXPERIMENTAL RESULTS AND DISCUSSION

##### A. Measured Performance

A prototype *Kerneltron* was integrated on a  $3 \times 3$  mm<sup>2</sup> die and fabricated in 0.5  $\mu$ m CMOS technology. The chip contains an array of  $256 \times 128$  CID/DRAM cells, and a row-parallel bank of 128 algorithmic  $\Delta\Sigma$  ADCs. Fig. 3(b) depicts the micrograph and system floorplan of the chip.

The processor interfaces externally in digital format. Two separate shift registers load the templates (support vectors) along odd and even columns of the DRAM array. Integrated refresh circuitry periodically updates the charge stored in the array to compensate for leakage. Vertical bit lines extend across the array, with two rows of sense amplifiers at the top and bottom of the array. The refresh alternates between even and odd columns, with separate select lines. Stored charge corresponding to matrix element values can also be read and shifted out from the chip for test purposes. All of the supporting digital clocks and control signals are generated on-chip.

Fig. 4(b) shows the measured linearity of the computational array, configured differentially for signed (XOR) multiplication. The case shown is where all complementary weight storage elements are actively set, and an alternating sequence of bits in blocks  $N$  is shifted through the input register.<sup>4</sup> For every shift in the input register, a computation is performed and the result is observed on the output sense line. The array dissipates 3.3 mW for a 10  $\mu$ s cycle time. The bank of  $\Delta\Sigma$  ADCs dissipates 2.6 mW yielding a combined conversion rate of 12.8 Msamples/s. Table I summarizes the measured performance.

<sup>4</sup> $w_{mn}^{(i)} = 1$ ;  $x_n^{(j)} = -1$  for  $n = 1, \dots, N$ ; and  $x_n^{(j)} = 1$  for  $n = N + 1, \dots, 2N$ .

TABLE I  
SUMMARY OF CHARACTERISTICS

Technology	0.5 $\mu\text{m}$ CMOS
Area	3mm $\times$ 3mm
Power	5.9 mW
Supply Voltage	5 V
Dimensions	256 inputs $\times$ 128 templates
Throughput	6.5 GMACS
Output Resolution	8-bit

### B. System-Level Performance

Fig. 7 compares template matching performed by a floating point processor and by the Kerneltron, illustrating the effect of quantization and limited precision in the analog array architecture. An “eye” template was selected as a  $16 \times 16$  fragment from the *Lena* image, yielding a 256-dimensional vector. Fig. 7(c) depicts the two-dimensional convolution (inner-products over a sliding window) of the 8-bit image with the 8-bit template computed with full precision. The same computation performed by the Kerneltron, with 4-bit quantization of the image and template and 8-bit quantization of the output, is given in Fig. 7(d). Differences are relatively small, and both methods return peak inner-product values (top matches) at both eye locations in the image.<sup>5</sup> The template matching operation is representative of a support vector machine that combines nonlinearly transformed inner-products to identify patterns of interest.

### C. Large-Scale Computation

The design is fully scalable, and can be expanded to any number of input features and support vectors internally as limited by current fabrication technology, and externally by tiling chips in parallel.

The dense CID/DRAM multiply-and-accumulate cell ( $18\lambda \times 45\lambda$ , where  $\lambda$  is the technology scaling parameter) supports the integration of millions of cells on a single chip in deep sub-micron technology, for thousands of support vectors in thousand dimensional input space as the line-width of the fabrication technology continues to shrink. The quantizer area overhead is less than 75% and becomes insignificant with larger array sizes for the same output resolution. In 0.18  $\mu\text{m}$  CMOS technology (with  $\lambda = 0.1 \mu\text{m}$ ), 64 computational arrays with  $256 \times 128$  cells each can be tiled on a 8 mm  $\times$  8 mm silicon area, with two million cells integrated on a single chip.

Distribution of memory and processing elements in a fine-grain multiply-and-accumulate architecture, with local bit-parallel storage of the  $W_{mn}$  coefficients, avoids the memory bandwidth problem that plagues the performance of CPUs and DSPs. Because of fine-grain parallelism, both *throughput* and *power dissipation* scale linearly with the number of integrated elements, so every cell contributes one kernel unit operation and one fixed unit of dissipated energy per computational cycle. Let us assume a conservative cycle time of 10  $\mu\text{s}$ . With two million cells, this gives a computational throughput of

<sup>5</sup>The template acts as a spatial filter on the image, leaking through spectral components of the image at the output. The *Lena* image was mean-subtracted.

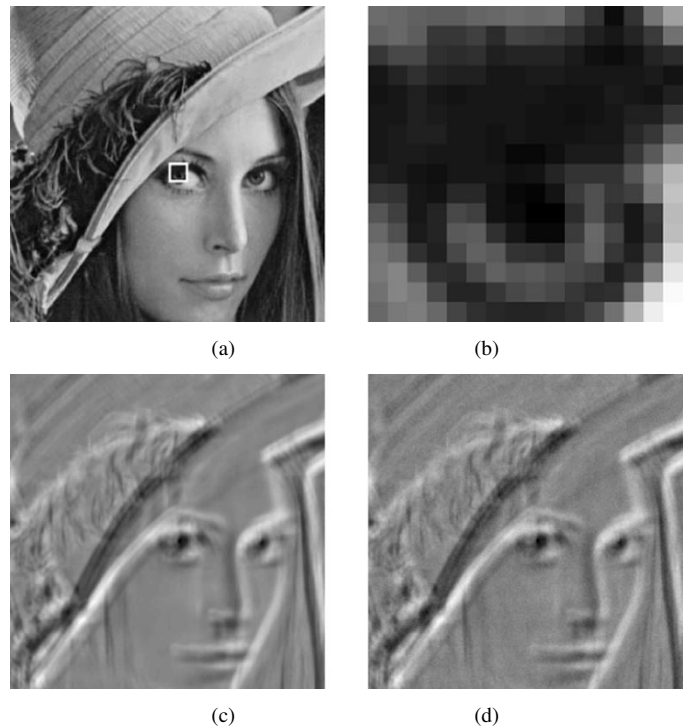


Fig. 7. Convolution of fragments of *Lena* (a) and the eye template (b) computed by a 32-bit floating point processor with 8-bit encoded inputs (c) and by Kerneltron with 8-bit quantization and 4-bit encoded inputs (d).

200 GOPS, which is adequate for the task described in Section II-B. The (dynamic) power dissipation is estimated<sup>6</sup> to be less than 50 mW which is significantly lower than that of a CPU or DSP processor even though computational throughput is many orders of magnitude higher.

### D. Training

Training of a support vector machine entails a quadratic programming problem of dimensions square in the number of data points. In principle, the training can be formulated as a constrained Hopfield neural network, with a natural analog circuit implementation [25]. The problem with this approach is that the area of the implementation scales with the square of the number of data points, which becomes impractical for very large data sets or a real-time (online) setting.

The incremental SVM learning approach in [12] allows on-line training of the Kerneltron with minimal overhead in implementation resources. Every misclassified training vector is stored as a (margin or error) support vector  $\mathbf{X}^m$  in the array, and the corresponding (nonzero) coefficient  $\alpha_m$  is computed using a recursive matrix operation of dimensions square in the number of margin (not error) support vectors. Since the number of margin vectors is usually very small compared with the number of training vectors, computational savings can be significant. Recursive computation of the coefficient  $\alpha_m$  is conveniently implemented off-chip, using the inner-products  $\mathbf{X}^m \cdot \mathbf{X}^i$  computed efficiently on the array.

<sup>6</sup>The parameters of the estimate are  $\lambda = 0.1 \mu\text{m}$ ; 3 V power supply; 10  $\mu\text{s}$  cycle time.

The limited number of templates  $X^m$  on the Kerneltron requires a trimming scheme to eliminate (most) inactive support vectors to make room for new support vectors. Estimations of the  $\alpha_m$  coefficients can further be simplified for integrated implementation as reported in [26]. A decomposition algorithm such as [27] or [28] offers an equally efficient realization in hardware, but requires multiple passes through the data for proper convergence of the coefficients.

### E. Applications

The *Kerneltron* benefits real-time applications of object detection and recognition, particularly in artificial vision and human-computer interfaces. Applications extend from SVMs to any pattern recognition architecture that relies on computing a kernel distance between an input and a large set of templates in large dimensions.

Besides throughput, power dissipation is a main concern in portable and mobile applications. Power efficiency can be traded for speed, and a reduced implementation of dimensions similar to the version of the pedestrian classifier running on a Pentium PC (27 input features) [4], [5] could be integrated on a chip running at 100  $\mu$ W of power, easily supported with a hearing aid type battery for a lifetime of several weeks.

One low-power application that could benefit a large group of users is a navigational aid for visually impaired people. OpenEyes, a system developed for this purpose [8] currently runs a classifier in software on a Pentium PC. The software solution offers great flexibility to the user and developer, but limits the mobility of the user. The *Kerneltron* offers the prospect of a low-weight, low-profile alternative.

## V. CONCLUSION

A massively parallel mixed-signal VLSI processor for kernel-based pattern recognition in very high dimensions has been presented. Besides support vector machines, the processor is capable of implementing other architectures that make intensive use of kernels or template matching. An internally analog, externally digital architecture offers the best of both worlds: the density and energetic efficiency of a charge-mode analog VLSI array, and the convenience and versatility of a digital interface.

An oversampling configuration relaxes precision requirements in the quantization while maintaining 8-bit effective output resolution, adequate for most vision tasks. Higher resolution, if desired, can be obtained through stochastic encoding of the digital inputs [24].

A  $256 \times 128$  cell prototype was fabricated in 0.5  $\mu$ m CMOS. The combination of analog array processing, oversampled input encoding, and delta-sigma analog-to-digital conversion yields a computational throughput of over 1 GMACS per milliwatt of power. The architecture is scalable and capable of delivering 200 GOPS at 50 mW of power in a 0.18  $\mu$ m technology—a level of throughput and efficiency suitable for real-time SVM detection of complex objects on a portable platform.

## REFERENCES

- [1] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifier," in *Proc. 5th Annu. ACM Workshop Computational Learning Theory*, 1992, pp. 144–52.
- [2] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.
- [3] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," *Comput. Vis. Pattern Recogn.*, pp. 130–136, 1997.
- [4] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," *Comput. Vis. Pattern Recogn.*, pp. 193–199, 1997.
- [5] C. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Int. Conf. Computer Vision ICCV*, 1998.
- [6] H. Sahbi, D. Geman, and N. Boujemaa, "Face detection using coarse-to-fine support vector classifiers," presented at the IEEE Int. Conf. Image Processing, Rochester, NY, 2002.
- [7] R. Genov, "Massively Parallel Mixed-Signal VLSI Kernel Machines," Ph.D. dissertation, Johns Hopkins Univ., Baltimore, MD, 2002.
- [8] S. Kang and S.-W. Lee, "Handheld computer vision system for the visually impaired," in *Proc. 3rd Int. Workshop Human-Friendly Welfare Robotic Syst.*, Daejeon, Korea, 2002, pp. 43–48.
- [9] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.*, vol. 7, pp. 219–269, 1995.
- [10] M. Pontil and A. Verri, "Properties of support vector machines," *Neural Comput.*, vol. 10, pp. 977–996, 1998.
- [11] C. Burges, "A tutorial on support vector machines for pattern recognition," in *Proc. Data Mining Knowledge Discovery*, U. Fayyad, Ed., 1998, pp. 1–43.
- [12] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Proc. IEEE Advanced Neural Information Processing Systems, Conf., NIPS*, 2001.
- [13] G. Cauwenberghs and M. Bayoumi, *Learning on Silicon, Analog VLSI Adaptive Systems*. Norwell, MA: Kluwer, 1999.
- [14] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with Gaussian kernels to radial basis functions classifiers," *IEEE Trans. Signal Processing*, vol. 45, pp. 2758–2765, Nov. 1997.
- [15] A. Kramer, "Array-based analog computation," *IEEE Micro Mag.*, vol. 16, pp. 40–49, May 1996.
- [16] A. Chiang, "A programmable CCD signal processor," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1510–1517, Nov. 1990.
- [17] C. Neugebauer and A. Yariv, "A parallel analog CCD/CMOS neural network IC," in *Proc. IEEE Int. Joint Conf. Neural Networks*, vol. 1, Seattle, WA, 1991, pp. 447–451.
- [18] V. Pedroni, A. Agranat, C. Neugebauer, and A. Yariv, "Pattern matching and parallel processing with CCD technology," *Proc. IEEE Int. Joint Conf. Neural Networks*, vol. 3, pp. 620–623, 1992.
- [19] M. Howes and D. Morgan, Eds., *Charge-Coupled Devices and Systems*. New York: Wiley, 1979.
- [20] R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for matrix-vector multiplication," *IEEE Trans. Circuits Syst. II*, vol. 48, pp. 930–936, Oct. 2001.
- [21] R. Harjani and T. A. Lee, "FRC: A method for extending the resolution of Nyquist rate converters using oversampling," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 482–494, Apr. 1998.
- [22] O. J. A. P. Nys and E. Dijkstra, "On configurable oversampled A/D converters," *IEEE J. Solid-State Circuits*, vol. 28, pp. 736–742, July 1993.
- [23] G. Mulliken, F. Adil, G. Cauwenberghs, and R. Genov, "Delta-sigma algorithmic analog-to-digital conversion," in *Proc. IEEE Int. Symp. Circuits Systems*, Scottsdale, AZ, May 26–29, 2002.
- [24] R. Genov and G. Cauwenberghs, "Stochastic Mixed-Signal VLSI Architecture for High-Dimensional Kernel Machines," in *Proc. IEEE Advanced Neural Information Processing Systems, Conf., NIPS*, 2002.
- [25] D. Anguita, S. Ridella, and S. Rovetta, "Circuitual implementation of support vector machines," *Electron. Lett.*, vol. 34, no. 16, 1998.
- [26] R. Genov, S. Chakrabarty, and G. Cauwenberghs, "Silicon support vector machine with on-line learning," *Int. J. Pattern Recognition Artificial Intell.*, vol. 17(3), pp. 385–404, 2003.
- [27] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.
- [28] S. Keerthi, C. Bhattacharyya, and K. Murthy, "A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design," Dept. CSA, IISc, Bangalore, India, Tech. Rep. TR-ISL-99-03, 1999.





**Roman Genov** (S'96–M'02) received the B.S. degree with highest honors in electrical engineering from Rochester Institute of Technology, Rochester, NY, in 1996 and the M.S. and Ph.D. degrees in electrical and computer engineering from the Johns Hopkins University, Baltimore, MD, in 1998 and 2002, respectively.

He held engineering positions at Atmel Corporation, Columbia, MD, in 1995 and Xerox Corporation, Rochester, NY, in 1996. He was a Visiting Researcher with the Robot Learning Group at Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland in 1998 and in the Center for Biological and Computational Learning at Massachusetts Institute of Technology, Cambridge, in 1999. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. His research interests include analog and digital VLSI circuits, systems and algorithms for parallel signal processing and adaptive neural computation with application to pattern recognition, focal-plane imaging, autonomous system design, and low-power instrumentation.

Dr. Genov is a Member of the IEEE Circuits and Systems Society and Solid-State Circuits Society. He received a Best Presentation Award at IEEE IJCNN'2000 and a Student Paper Contest Award at IEEE MWSCAS'2000.



**Gert Cauwenberghs** (S'89–M'94–SM'03) received the Ph.D. degree in electrical engineering from California Institute of Technology, Pasadena, in 1994.

He is presently Professor of Electrical and Computer Engineering at the Johns Hopkins University, Baltimore, MD. He was a Visiting Professor of Brain and Cognitive Science at Massachusetts Institute of Technology, Cambridge, in 1999. His research covers VLSI circuits, systems and algorithms for parallel signal processing, adaptive neural computation, and low-power coding and instrumentation.

He recently co-edited a book on *Learning on Silicon* (Norwell, MA: Kluwer, 1999).

Dr. Cauwenberghs was a Francqui Fellow of the Belgian American Educational Foundation in 1988, and received the National Science Foundation Career Award in 1997, the Office of Naval Research Young Investigator Award in 1999, and the Presidential Early Career Award for Scientists and Engineers in 2000. He is Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II, and the IEEE SENSORS JOURNAL. He was Publication Chair at IEEE MWSCAS'2000, Analog Track Chair at IEEE ISCAS'2002, and Invited Sessions Co-Chair at ISCAS 2003. He chaired the Analog Signal Processing Technical Committee of the IEEE Circuits and Systems Society in 2001.