A Fast Algorithm for Computing the Euler Number of an Image and its VLSI Implementation*

Sabyasachi Dey , Bhargab B. Bhattacharya†, Malay K. Kundu and Tinku Acharya‡
Indian Statistical Institute
203 B. T. Road, Calcutta - 700 035, India
{mtc9701, bhargab, malay}@isical.ac.in
‡Digital Imaging and Video Division
Intel Corporation, Chandler, AZ 85226, USA
tinku.acharya@intel.com

Abstract

Digital images are convenient media for describing and storing spatial, temporal, spectral, and physical components of information contained in a variety of domains(e.g. aerial/satellite images in remote sensing, medical images in telemedicine, fingerprints in forensics, museum collections in art history, and registration of trademarks and logos). Euler number is a fundamental topological feature of an image. The efficiency of computation of topological features of an image is critical for many digital imaging applications including image matching, database retrieval, and computer vision, that require real time response. In this paper, a novel algorithm for computing the Euler number of a binary image based on divide-and-conquer paradigm, is proposed, which outperforms significantly the conventional techniques used in image processing tools. The algorithm can be easily parallelized for computing the Euler number of an N×N image in $O(N)$ time, with $O(N)$ processors. Using a simple architecture, the proposed method can be implemented as a special purpose VLSI chip.
Index Terms - Digital imaging, Euler number, VLSI, parallel processing, binary image.

1 Introduction

Topological properties serve the purpose of representing geometric shape of an image. They remain invariant under any arbitrary *rubber-sheet* transformation [2, 3, 7] and hence, are very useful in image characterization for matching shapes, recognizing objects, image database retrieval, and many other numerous image processing and computer vision applications. An important topological feature of an image is the Euler number (or genus), which is the difference between the number of connected components and the number of holes [1, 2].

With the emergence of fast internet facilities, a growing demand for distributed and high-performance image retrieval systems is being felt. Euler number can play an important role in such applications. It can also be used in medical diagnosis from cell images, e.g., detection of malaria infected cells, as the Euler number of an infected cell is often different from that of a good one. Critical image processing applications involve large amount of data and at the same time demands for real-time response. Fast computation of the Euler number of an image is therefore, an indispensable task in various missions.

Dyer proposed an algorithm to compute the Euler number of an image represented by a quadtree [4]. Samet and Tamminen improved the algorithm further by using a new staircase type of data structure to represent the blocks that have already been processed [5]. However, it is not suitable for VLSI implementation, as the sizes of the leaf nodes are unequal, and the number of leaf nodes varies widely for different image samples. Pratt [7] has described a method based on local pattern counting which is used in commercial image processing tools like MATLAB [8]. The time complexity of this method is $O(N^2)$ for an N×N image. For large images, even a quadratic algorithm may be inadequate to meet critical time requirements. A faster algorithm is thus needed to handle large binary images. Recent advances in parallel processing and VLSI technology can be ex-

*This work is funded by a grant from Intel Corp., USA (PO # CAC042717000)
†Author for correspondence.

ploited to develop high performance algorithm and architecture that achieves real-time goals. In this paper, a new algorithm that computes the Euler number of a binary image is proposed, based on divide-and-conquer approach. Although the worst case complexity of the algorithm is $O(N^2)$, its average-case behavior outperforms the earlier algorithms significantly. Next, a parallel version of the algorithm is described that takes $O(N)$ time for an N×N image, using $O(N)$ simple processors, which can be readily mapped to a simple VLSI architecture. No hardware for efficient computation of Euler number seems to have been designed till date. The proposed algorithm can easily be implemented with a special purpose VLSI chip that can serve as a co-processor to the host computer to expedite computation.

2 Preliminaries

The most efficient and simplest algorithm, reported so far, for computing Euler number of an image works by looking into local patterns [7]. Consider the following set of 2×2-pixel patterns called bit quad:

$$\mathbf{Q_1} = \left\{ \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}, \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}, \begin{matrix} 0 & 1 \\ 0 & 0 \end{matrix}, \begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix} \right\}$$

$$\mathbf{Q_2} = \left\{ \begin{matrix} 0 & 1 \\ 1 & 1 \end{matrix}, \begin{matrix} 1 & 0 \\ 1 & 1 \end{matrix}, \begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}, \begin{matrix} 1 & 1 \\ 0 & 1 \end{matrix} \right\}$$

$$\mathbf{Q_3} = \left\{ \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}, \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \right\}$$

Let C_1, C_2 and C_3 be the number of patterns Q_1, Q_2, Q_3 respectively in the image S. It has been shown that under the definition of four-connectivity the Euler number can be computed as

$$E(S) = \frac{1}{4}\left(C_1 - C_2 + 2 \cdot C_3\right) \tag{1}$$

and for eight-connectivity

$$E(S) = \frac{1}{4}\left(C_1 - C_2 - 2 \cdot C_3\right) \tag{2}$$

This method is used in MATLAB image processing tool box [8]. Henceforth, we shall refer this method as Pratt algorithm.

In the proposed divide-and-conquer approach, the input image is partitioned into a number of disjoint atomic images. By atomic is meant a small part of the input image which need not be decomposed further. The Euler number of each atomic image, which is substantially smaller in size compared to the input image, can be computed by the method described above. Once the Euler numbers of all atomic images are evaluated, the Euler number of the original image can be computed by using a simple arithmetic rule. The given image is partitioned recursively by arbitrary cut-lines. A cut line is a sequence of pixels from one boundary of the image to its opposite boundary, where each pixel has exactly two neighboring pixels along the cut-line (except the start and end pixels which have only one neighbor each). Without any loss of generality, we restrict the choice of cut-lines to only horizontal and vertical directions. Since a binary image is normally represented as a 2-D $(0-1)$ pixel matrix, any row or column of the matrix can be used to designate a cut-line. A *run* on a cut-line is defined to be a maximal sequence of 1-pixels.

3 Proposed Algorithm

3.1 Divide-and-conquer approach

Let S be the given binary image, and L denotes a cut-line that partitions S into two sub-images S_1 and S_2. Let ρ be a run on line L. Denote by $k_1(\rho)$ the number of runs on the boundary line of S_1 that are adjacent to (neighbor of) ρ, and by $k_2(\rho)$, the number of such runs in S_2.

Theorem 1 *If a binary image S is partitioned into two sub-images S_1 and S_2 along a cut-line L such that,*

1. $S = S_1 \cup S_2 \cup L$, *and*
2. $S_1 \cap S_2 \cap L = \phi$

then, the Euler number E(S) of the image S is given by

$$E(S) = E(S_1) + E(S_2) + Cont(L) \tag{3}$$

where,
$Cont(L) = \sum_{\rho \in L}\{1 - k_1(\rho) - k_2(\rho)\}$. □

The proof of the above theorem is given in Appendix A. Here, we illustrate the idea with an example. Image S (see figure 1) is partitioned into S_1 and S_2 by cut L_1. The component S_1 is again partitioned into S_{11} and S_{12} by cut L_2. Similarly, S_2 is partitioned into S_{21} and S_{22} by cut L_3.
$E(S_1) = E(S_{11}) + E(S_{12}) + Cont(L_2)$ where $Cont(L_2) = \sum_{r \in L_2}\{1 - k_1(\rho) - k_2(\rho)\} = 0$, and $E(S_{11}) = E(S_{12}) = 1$. Hence, $E(S_1) = 1+1+0 = 2$.
$E(S_2) = E(S_{21}) + E(S_{22}) + Cont(L_3)$ where $Cont(L_3) = \sum_{r \in L_3}\{1-k_1(\rho)-k_2(\rho)\} = 1-1-1 =$

-1,and $E(S_{21}) = E(S_{22}) = 1$. Hence, $E(S_2) = 1+1-1 = 1$.
$E(S) = E(S_1) + E(S_2) + Cont(L_1)$ where $Cont(L_1) = \sum_{\rho \in L_1}\{1-k_1(\rho)-k_2(\rho)\} = 3-3-3 = -3$,and $E(S_1) = 2$, and $E(S_2) = 1$. Hence, $E(S) = 2+1-3 = 0$.

In the next subsection, we describe a sequential algorithm based on recursive application of the above partitioning scheme.

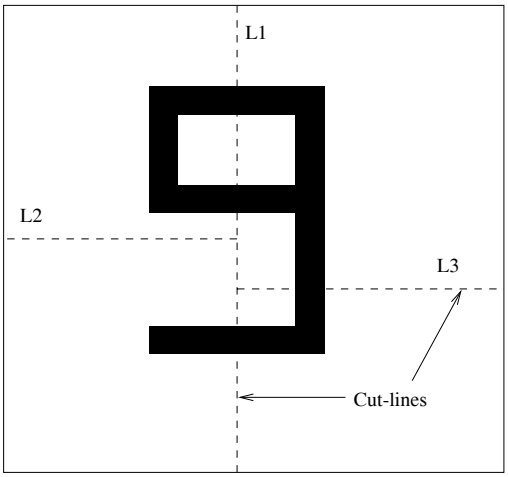


Figure 1: Divide-and-conquer illustrated

3.2 Sequential Algorithm

The recursive procedure *ComputeEuler* computes the Euler number of an image. If both the dimensions of the partitioned image become smaller than some predefined value MIN, then we compute its Euler number by procedure *Euler*. Procedure *Contribution* computes contribution of the cut-line. We shall show how to determine the value MIN after analyzing the parallel algorithm in section 4.3.

```
procedure ComputeEuler(S,hlen,vlen)
/*
      S is the input binary image,
      hlen is horizontal length of the image,
      vlen is vertical length of the image.
*/

begin
   if ( hlen < MIN and vlen < MIN ) then
   begin
       e = Euler(S);
   end
   else
   begin
    /*
      S is partitioned by cut-line
      L into S1 and S2
    */
    if ( hlen > vlen ) then
      begin
         x = ComputeEuler(S1,hlen/2,vlen);
         y = ComputeEuler(S2,hlen/2,vlen);
         z = Contrib(L);
         e = x + y + z;
      end
      else
      begin
         x = ComputeEuler(S1,hlen,vlen/2);
         y = ComputeEuler(S2,hlen,vlen/2);
         z = Contrib(L);
         e = x + y + z;
      end
   end
   return e;
end.
```

Procedure *Euler* computes Euler number of the atomic image by the method described in Pratt [7]. The basis of procedure *Contrib* is described below.

3.3 Contribution computation

In this section, we describe a method for computing contribution of the line L that partitions the image S into two sub-images S_1 and S_2 as mentioned in equation (3). Recall,

$$Cont(L) = \sum_{\rho \in L}\{1 - k_1(\rho) - k_2(\rho)\}$$

We thus have to count the number of runs along L, as well as number of runs along the boundary lines of S_1 and S_2 that are adjacent to runs on L. Consider the following 2×3 bit pattern.

$$\begin{pmatrix} x & u \\ y & v \\ z & w \end{pmatrix}$$

where

- $x, u \in S_1$,
- $z, w \in S_2$, and
- $y, v \in L$.

Without any loss of generality, we can assume that L is the i^{th} row of the image matrix S. Then S(i-1,j-1) = x, S(i-1,j) = u, S(i+1,j-1) = z, S(i+1,j) = w, S(i,j-1) = y, and S(i,j) = v. We traverse the band of three consecutive rows (i-1, i, and i+1)

column by column remembering the elements of the last column. Hence, when we look into column j, the elements of (j-1)th and jth columns are known. Let the number of runs on L be α, $\sum_\rho k_1(\rho) = \beta$, and $\sum_\rho k_2(\rho) = \gamma$. Since number of runs on L and number of adjacent runs in either side do not depend on each other we can write,

$$\sum_\rho \{1 - k_1(\rho) - k_2(\rho)\} = \alpha - \beta - \gamma$$

We define three boolean functions Γ , Ψ , and Φ. We increase α, β, and γ as following,

- if Γ is true then increment β by one,
- if Ψ is true then increment γ by one, and
- if Φ is true then increment α by one.

Under the definition of four-connectivity, the above boolean functions are defined as,

$$\Phi = \bar{y} \cdot v \tag{4}$$
$$\Gamma = u \cdot v \cdot (\bar{x} + \bar{y}) \tag{5}$$
$$\Psi = w \cdot v \cdot (\bar{z} + \bar{y}) \tag{6}$$

and for eight-connectivity they are defined as,

$$\Phi = \overline{y} \cdot v \tag{7}$$
$$\Gamma = \bar{x} \cdot u \cdot v + \bar{x} \cdot u \cdot y + x \cdot \bar{y} \cdot v \tag{8}$$
$$\Psi = \bar{z} \cdot w \cdot v + \bar{z} \cdot w \cdot y + z \cdot \bar{y} \cdot v \tag{9}$$

Thus, using the above equations, $Cont(L)$ can be evaluated easily from the pixel matrix.

3.4 Time Complexity

Although the asymptotic time complexity of procedure *ComputeEuler* is $O(N^2)$ which is same as that of the method described in [7] for a N×N image, its average case performance is much better, as we have a smaller constant factor. In the Pratt algorithm, each pixel of the image is accessed twice except the boundary pixels which are accessed only once. On the other hand, in the proposed divide-and-conquer method, each pixel along a cut-line is accessed only once, and pixels in the sub-images which are adjacent to 0's in the cut-line, need not be checked. Thus, the number of pixel accesses is significantly reduced on the average and consequently, we save time as we need fewer memory references. Experimental results of our implementation demonstrate the savings in computation time, and are shown below.

3.5 Experimental Results

The algorithm has been implemented in C and the code runs on both Solaris 2.6 and Linux platforms. We have tested the program with several hundred images out of which 10 examples are tabulated below. We compared our algorithm with the most efficient implementation of Pratt algorithm. The results are shown below. The columns labeled "Pratt" and "Proposed method" correspond to the number of pixel accesses, and the resulting improvement is given in percentage. For the Pratt method, a pixel is accessed twice but the boundary pixels are accessed only once, for determining convexity. Thus, it becomes slightly less than N×M×2 for an N×M image [10].

A. List of test images

Image name	Size	Euler number
tm1	128 x 128	-2
tm2	256 x 256	-2
tm3	256 x 256	-3
tm4	256 x 256	1
tm5	128 x 128	-9
tm6	128 x 128	0
ieee1	128 x 128	-1
ieee2	128 x 128	7
vlsi	512 x 512	9
text	256 x 256	6

B. Comparison with the most efficient Pratt implementation

Img	Pratt	Prop. meth.	Improvement
tm1	33024	28008	15.19
tm2	131584	104082	20.90
tm3	131584	109086	17.10
tm4	131584	111250	15.45
tm5	33024	27426	16.95
tm6	33024	28598	13.40
ieee1	33024	29086	14.92
ieee2	33024	25705	22.15
vlsi	525312	448774	14.57
text	131584	102880	22.77

4 Parallel Implementation

4.1 The Algorithm

In the proposed approach, each cut increases the number of sub-images by one. Let there be K atomic images and K-1 cut-lines. The parallel architecture that lends itself most naturally to this divide-and-conquer algorithm is a (binary) tree. We use two types of processing elements

(PE's). Type-1 PE computes procedure *Contrib* and evaluates equation (3). Type-2 PE computes the Euler number of an atomic image, i.e., computes procedure *Euler*. The PE's are organized in a binary tree, in which the type-2 PE's are leaf nodes and the type-1 PE's are non-leaf nodes. There are K-1 type-1 PE's denoted by $PE_{1,1}, PE_{1,2}, \ldots, PE_{1,K-1}$ and K type-2 PE's denoted by $PE_{2,K}, PE_{2,K+1}, \ldots, PE_{2,2K-1}$. Each type-1 PE, $PE_{1,i}$ where $i \neq 1$ is connected to $PE_{1,i/2}$ if i is even and to $PE_{1,(i-1)/2}$ if i is odd. Each type-2 PE, $PE_{2,j}$ is connected to $PE_{2,j/2}$ if j even and to $PE_{2,(j-1)/2}$ if j odd. Let the cut-lines be $L_1, L_2, \ldots, L_{K-1}$. The contribution of L_i is computed by $PE_{1,i}$. Let the atomic images be $S_1, S_2, \ldots, S_K$, where the Euler number of S_j is computed by $PE_{2,j+k-1}$. The atomic images S_j and S_{j+1}, where j is 1, 3, ..., K-1, have been generated by the cut-line $L_{(j+k-1)/2}$. If L_i partitions an image S into S_1 and S_2 then all computations for S_1 are done in left sub-tree of node $PE_{1,i}$, and those for S_2 are done in right sub-tree of node $PE_{1,i}$. Let h be the height of the tree.

SIMD SM CREW Algorithm

Step 1a : for i = 1 to K-1 do in parallel
$PE_{1,i}$ computes contribution of line L_i;
endfor.

Step 1b : for i = 1 to K do in parallel
$PE_{2,i}$ computes $E(S_i)$;
$PE_{2,i}$ sends the result to its parent;
endfor.

Step 2 : for l = h-1 downto 2 do
for i = 2^{l-1} to $2^l - 1$ do in parallel
$PE_{1,i}$ evaluates eq.(3);
$PE_{1,i}$ sends the result to its parent;
endfor.
endfor.
$PE_{1,1}$ evaluates eq.(3) and outputs result.

4.2 VLSI Architecture

How the PE's are organized in a binary tree has been described before. Organization of the PE's is depicted in Figure 2. In this section we sketch the internal organization of each PE. Corresponding circuit diagrams are given in Appendix C.

A. Type-1 processing element

The circuit for computing the contribution of a cut-line is very simple. It consists of three D flip-flop's FF_1, FF_2, FF_3 with inputs u, v, w and outputs x, y, z respectively. In each clock cycle, a new column of the 3-pixel wide band is fed to the inputs of the FF's and the values for the earlier column are already at their outputs. According to the equations in Section 3.3 a combinatorial circuit consisting of basic gates evaluates Γ, Ψ, and Φ. In Appendix C the combinational blocks for computing $\Gamma, \Psi, and \Phi$ are shown as a triangular object. We use three counters C_1, C_2, and C_3 driven by the values of Γ, Φ, Ψ respectively. Contribution is computed as,

$$Cont(L) = C_2 - C_1 - C_3.$$

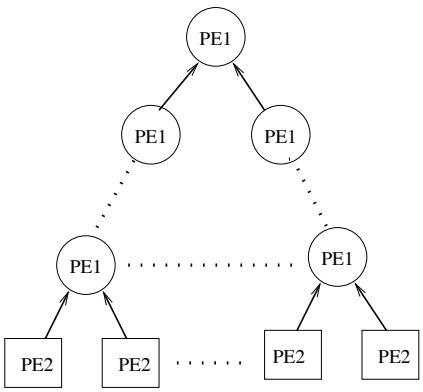


Figure 2: Processor Organization in a Tree

B. Type-2 processing element

We design a simple hardware that computes the Euler number by counting number of patterns Q_1, Q_2, Q_3 (see section 2) in the image S. Note that at any point we are looking into only four pixels. For binary images each pixel is nothing but a bit. If we label the pixels from the top-left corner in clockwise fashion as $b0, b1, b2, and b3$ then we get a bit string of length four where the label of the pixel serves as its position in the string. The idea is illustrated below.

$b0$	$b1$
$b3$	$b2$

The bit string is then fed to a 4×16 decoder. The output lines of the decoder are $d_0, d_1, \ldots, d_{15}$. Three counters C_1, C_2, C_3 count the number of patterns Q_1, Q_2, Q_3 respectively (see fig.). The counters are set to zero initially and incremented as follows,

- C_1 is incremented when line d_1, d_2, d_4, or d_8 is set.
- C_2 is incremented when line d_7, d_{11}, d_{13}, or d_{14} is set.

- C_3 is incremented when line d_5 or d_{10} is set.

When all the pixels are visited, the Euler number is computed from the counter values by using equations (1) and (2).

4.3 Computational Complexity

The proposed architecture requires 2K-1 PE's for an input image of size N×N which is partitioned into K atomic images. Let the maximum size of an atomic image be n×m ($n < m$) where

$$N^2 > knm \tag{10}$$

By incorporating a memory prefetch buffer, data can be supplied to the PE's in a single clock cycle. In step 1, all the PE's are active. The time needed for executing step 1 is the maximum time taken by a PE to compute procedure *Euler* of *Contrib*. In a CREW model each type-1 PE needs

1. 1 clock cycle to read three pixel data,
2. 1 clock cycle for the combinational circuit, and
3. 1 clock cycle to drive the counters.

Moreover, these three operations can be executed in a three-stage pipeline. Hence, each type-1 PE takes P clock cycles where P is the length of the cut-line which is at the most N. Since Step 1a in the algorithm is executed in parallel, computation of contribution takes only N clock cycles.
Each type-2 PE needs

1. 1 clock cycle to read pixel data,
2. 1 clock cycle to decode and evaluate the Boolean functions, and
3. 1 clock cycle to drive the counters.

Similarly these three operations also can be executed in a three-stage pipeline. As step 1b in the algorithm is executed in parallel, computation of the Euler number of all the atomic images takes n^2 clock cycles by the type-2 PE's. If we make $n^2 < N$,the overall time requirement for step 1 becomes N. Step 2 is executed for $h = \log_2 K$ times. Now from eqn.(10) we have $K < \frac{N^2}{nm}$. Hence time requirement for step 2 is $O(\log_2 N)$, and the total computation needs $N + O(\log_2 N)$ clock cycles. Thus, time complexity of the parallel algorithm is $O(N)$. As the number of PE's is $O(N)$ and computation time is $O(N)$, we achieve a speed-up of $O(N)$ for an N×N input image. The AT value [9] is $O(N^2)$ which is optimum as the sequential algorithm for computing the Euler number of an binary image takes $O(N^2)$ time for an N×N image [8]. The constraint $n^2 < N$ determines the value of MIN as mentioned in Section 3.2.

5 Conclusion

We have presented a fast recursive algorithm for computing the Euler number of a binary image, that provides a significant savings in computation time. A parallel version of the algorithm is then described that runs in $O(N)$ time on a tree architecture with $O(N)$ simple processing elements, for an N×N image. The AT-measure of the implementation is $O(N^2)$ which is optimum. A hardware implementation of the scheme has been proposed. Since the architecture needs simple interconnections among only two types of processing elements, it is highly suitable for single-chip VLSI implementation.

Acknowledgment

We thank Dr. Anil K. Jain and Aditya Vailya of Michigan State University, U.S.A. for providing us with the trademark database.

References

[1] M. Minsky and S. Papert, "Perceptrons : An Introduction to Computational Geometry," *The MIT Press*, 1969, p. 86.

[2] R. C. Gonzalez and R. E. Woods, "Digital Image Processing," *Addison-Wesley,* Reading, Massachusetts, 1993, pp. 504-506.

[3] E. C. Greanis et al.,"The Recognition of Handwritten Numerals by Contour Analysis,"*IBM J. Res. Dev.,* Vol. 7, No. 1, Jan. 1963, pp.14-21.

[4] C. R. Dyer,"Computing the Euler number of an image from its quadtree," *Comput. Graphics Image Processing,* Vol. 13, No. 3, pp.270-276, July 1980.

[5] H. Samet and M. Tamminen,"Computing Geometric Properties of Images Represented by Linear Quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.,* Vol. PAMI-7, No. 2, March 1985.

[6] S. B. Gray,"Local Properties of Binary Images in Two Dimension,"*IEEE Trans. Computers,* 2. 20, no. 5, May 1971, pp. 551-561.

[7] W. K. Pratt, "Digital Image Processing." *John Wiley & Sons.,* 1978.

[8] C. M. Thompson and L. Shure, "Image Processing Toolbox." *The Math Works Inc.*

[9] C. D. Thompson, "A complexity theory for VLSI." *Ph.D. dissertation,* Dept. of Comp. Sc., CMU, Aug. 1990.

[10] S. Dey," VLSI for Image Processing : A Case Study with Geometric Features." *M.Tech. (CS) Thesis,* Indian Statistical Institute, Calcutta 700 035, India, July 1999.

[11] A. K. Jain and A. Vailaya,"Shape-based Retrieval : A Case Study with Trademark Databases," *Pattern Recognition,* Vol. 31, No. 9,1998, pp. 1369-1390.

Appendix A

Proof of Theorem 1 : Consider two runs R_1 $(p_1, p_2, \ldots, p_n)$ and R_2 $(q_1, q_2, \ldots, q_m)$. They are said to be adjacent if and only if there exists at least one pixel $p_i \in R_1$ and one pixel $q_j \in R_2$ such that p_i is in the neighborhood of q_j and vice versa. Since, the runs along a cut-line are separated by 0's, their contributions to global connectivity are mutually exclusive. Hence, it is enough to show that the theorem holds for a single run. If the cut-line does not touch or divide any object then we can add the Euler number of two sub-images to get the Euler number of the original image. In this case $k_1(R) = k_2(R) = 0$. Assume that the cut-line L partitions the image S into S_1 and S_2. Let R be a run on L.

Case A. $k_1(R) = k_2(R) = 0$.
If $k_1(R) = k_2(R) = 0$ then R is a separate object in original object which is not connected to any other object either in S_1 or in S_2. Hence clearly, $E(S) = E(S_1) + E(S_2) + 1$, and $\sum_{R \in L}\{1 - k_1(R) - k_2(R)\} = 1$.

Case B. Either of $k_1(R)$ *and* $k_2(R)$ *is 1 and other is 0.*
Clearly, the Euler number of the sub-images are additive since R just touches boundary of one object which has already been counted in one of the sub-images. Hence we don't need to add or subtract anything from $E(S_1) + E(S_2)$. This conforms eqn.(3) since $1 - k_1(R) - k_2(R) = 0$.

Case C. $k_1(R) = k_2(R) = 1$.
Basically one object has been partitioned by the run R and the same object has been computed in both the sub-images. To nullify the effect we must subtract 1 from $E(S_1) + E(S_2)$. In this case $1 - k_1(R) - k_2(R) = -1$.

Case D. $k_1(R) = p, k_2(R) = q$.
If two runs in S_1 (or S_2) and connected in S_1 (or S_2), clearly, they create a hole along with R and thus effectively decrease the Euler number of S by 1. On the other hand, if they are not connected then they are counted as two separate objects where in S they are connected by R and hence must be treated as a single object. Thus in both the cases, regardless of whether two adjacent runs are connected, we have to subtract 1 from $E(S_1) + E(S_2)$. By induction on the number of adjacent runs, it can be shown that for p adjacent runs we must subtract $p - 1$. Further, the object which is partitioned by the cut-line, has been counted twice (once in each of the two sub-images), and so 1 should be subtracted from the count. Thus, $E(S) = E(S_1) + E(S_2) - (p-1) - (q-1) - 1$
$= E(S_1) + E(S_2) + (1 - p - q)$
$= E(S_1) + E(S_2) + 1 - k_1(R) - k_2(R)$.
Summing over all the runs, we get eqn. (3). Hence the theorem is proved. □

Appendix B : Test figures

Test figures are shown here. Trademark images have been taken from a database maintained by Dr. Anil Jain and Aditya Vailya at Michigan State University. IEEE and VLSI figures have been scanned from Proceedings of VLSI Design Conference 1999 using HP scanjet. Text figure has been created by xfig utility on Solaris platform. Figures are not shown in their original size. They have been normalized to same size.

Trademark 1

Trademark 2

Trademark 3

Trademark 4

Trademark 5

Trademark 6

IEEE 1

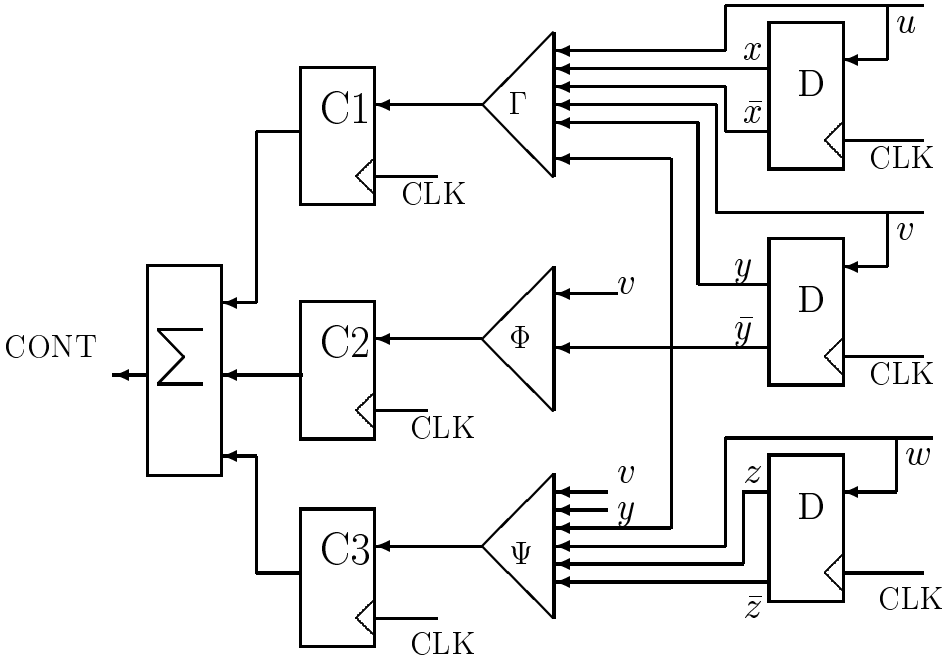
IEEE 2

VLSI

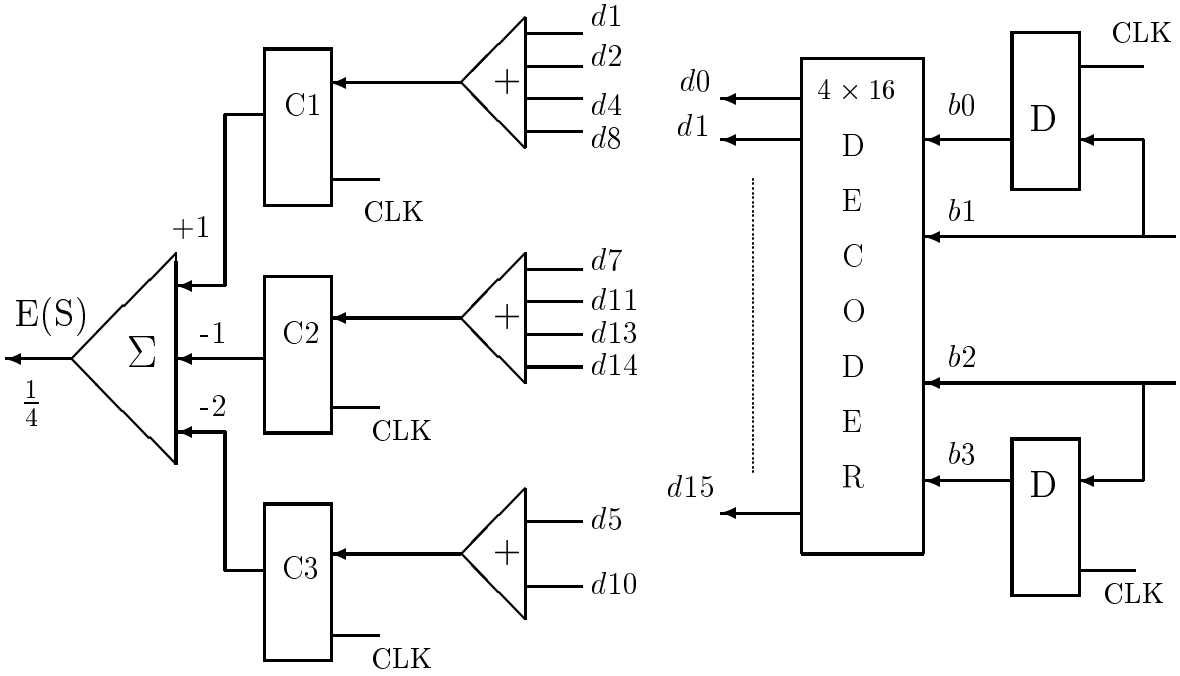
Hello
World

Text

Appendix C : PE Architecture



Architecture of Type 1 Processing element.



Architecture of Type 2 Processing element.