論文 87-24-6-18

MultiRing An Efficient Hardware Accelerator for Design Rule Checking

(멀티링 설계규칙검사를 위한 효과적인 하드웨어 가속기)

魚 吉 秀,* 慶 宗 旻*

(Kil-Su Eo and Chong-Min Kyung)

要 約

본 논문은 빠른 처리속도와 뛰어난 유연성을 가지는 VLSI 도면의 설계규칙검사와 잡음억제, 경계선도출을 포함하는 다양한 영상처리에 응용될 수 있는 하드웨어구조인 MultiRing을 소개한다. 그것의 전체적인 구성은 네가지 부분으로 이루어져 있는데, Host와 MultiRing사이의 입출력장치, ring memory, 일차원적 처리부 그리고 명령데코다가 그것들이다. 링 메모리에 저장된 data들은 column별로 병렬 처리된다. 각 프로세서는 명령데코다의 명령에 의해서 boolean연산(AND, OR, NOT, COPY), 수축과 확대, 감지 그리고 I/O동작등을 링사이클 단위로 수행하는데, MultiRing은 설계규칙의 변경이나 명령세트의 증가에 매우 뛰어난 유연성을 가진다. MultiRing의 하드웨어구조와 일치성을 갖는 소프트웨어 시뮬레이션을 통하여 MultiRing의 정확한 동작을 증명하였다.

Abstract

We propose a hardware architecture called MultiRing which is applicable for various geometrical operations on rectilinear objects such as design rule checking in VLSI layout and many image processing operations including noise suppression and contour extraction. It has both a fast execution speed and extremely high flexibility. The whole architecture is mainly divided into four parts ; I/O between host and MultiRing, ring memory, linear processor array and instruction decoder. Data transmission between host and MultiRing is bit serial thereby reducing the bandwidth requirement for the channel and the number of external pins, while each row data in the bit map stored in ring memory is processed in the corresponding processor in full parallelism. Each processor is simultaneously configured by the instruction decoder/controoler to perform one of the 16 basic instructions such as Boolean (AND, OR, NOT, and Copy), geometrical (Expand and Shrink), and I/O operations each ring cycle, which gives MultiRing maximal flexibility in terms of design rule change or the instruction set enhancement. Correct functional behavior of MultiRing was confirmed by successfully running a software simulator having one-to-one structural correspondence to the MultiRing hardware.

I. Introduction

Complexities and the required short turn-around of the recent VLSI circuits definitely require a very fast an flexible (in terms of

*正會員, 韓國科學技術院 電氣 및 電子工學科
(Dept. of Elec. Eng., KAIST)

接受日字 : 1986年 11月 24日

process parameter change) tools for design and verification. Conventional software approaches for various design automation problems on the serial machines already began to fall short of meeting this demand. Several special purpose hardware architectures tailored to processing two-dimensional geometrical problems including Design Rule Checking (DRC) in IC layout and wire routing have been reported to relieve the situation by exploiting hardware parallelism. [1], [2], [3] In this paper, we like to classify the hardware architectures for 2-D geometry processing reported so far into the following three categories according to the total number of unit processors involved for a 2-dimensional n x n data plane; 2-D class has $O(n^2)$ processors, while 1-D and O-D class has $O(n^1)$ and $O(n^0)$ processors, repectively.

Cytocomputer reported in [3] belongs to the O-dimensional class and is schematically shown in Fig. 1. Bit map data representing the 2-D geometry space is scanned out one pixel by one in row-major order and processed by a pipelined processor chain where each processor executes one of the basic operations into which the desired instruction is decomposed. Each processor in the pipeline stage itself consists of a subarray processor implemented as combinational logic circuitry and a shift register which has a length of at least two rows of the bit map plane to perform the neighborhood operations in the vertical direction as well as in the horizontal direction. In this architecture, however, the complexity of each processor becomes excessive because each processor has to contain a shift register array whose length is proportional to (length of two rows of the bit map plane) x (number of layers). Moreover, the structure of the unit processor and the number of stages in the processor pipeline is tied up to the specific bit map size and the specific set of operations. Also, its time complexity is $O(nm)$ for n x m bit map planes which is the slowest among the three classes.

On the other hand, there are reported several architectures [1], [2] belonging to the two-dimensional class which consists of two-dimensional array of the unit processor cells. [1] describes an array processor for two dimensional bit map data where the cells called SAM cell are linked in a two-dimensional mesh-connected network. Another example of this scheme is an interconnected array of microcomputers to solve the wire routing problem described in [2]. These schemes need minimal external data flow because bit data in each cell are directly accessible from its neighbors, resulting in high speed performance. But the complexity of this scheme is still prohibitive even with the present VLSI technology since the size of the bit map data could easily be more than 1000 x 1000 in VLSI layout. However, the time complexity is minimal, i.e., $O(1)$.

As a compromise between the slow speed of the 0-D class and the excessive hardware complexity of the 2-D class, we considered a 1-D class architecture as shown in Fig. 2. Each stage in this processor pipeline is a replica of the unit processor in the 0-D scheme as many times as the number of rows except that it needs no shift register array. Its time complexity is $O(n)$. However, since the number of stages of the processor pipelin is fixed, this architecture is not flexible for such changes in design rule parameter or instruction set

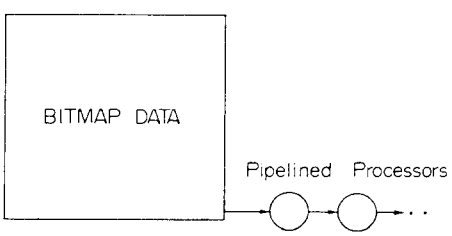


Fig. 1. 0-dimensional image processing scheme having $O(n^0)$ processing elements (PE's).

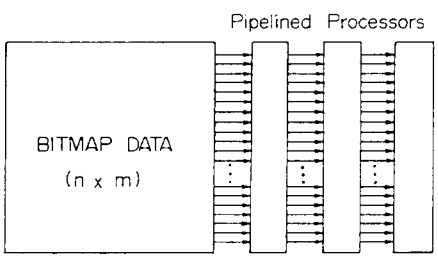


Fig. 2. 1-dimensional image processing scheme having $O(n^1)$ PE's.

enhancement. The property comparison between these schemes is presented in Table 1.

Table 1. Comparison of 4 schemes.

Schemes	Circuits	Speed	Flexibility
0-D	simple	low	low
1-D (pipeline)	middle	middle	low
1-D (MultiRing)	simple	middle	high
2-D	Complex	high	high

II. MultiRing-Hardware accelerator for DRC

1. Hardware architecture

An imaginary architecture for describing the operation of the proposed DRC hardware -MultiRing-is shown in Fig. 3. It consists of a multiple ring memory which corresponds to a foldback of the bit map plane in the 1-D class at its ends, and one stage of linear processor array. The bit map data stored in the ring memory circles around until necessary while the linear processor array executes a sequence of basic instructions which are externally selectable by a controller. The ring meory can be implemented as shift register array, as shown in Fig. 4. A possible candidate is dynamic shift register with a two-phase nonoverlapping clock where each memory cell consisting of two inverters and two pass transistors requires 8 transistors in CMOS technology and 6 in NMOS technology. (See Fig. 5)

A block diagram of the MultiRing is shown in Fig. 6. It consists of four parts, i.e., I/O, ring memory, linear processor array and instruction decoder. The I/O part handles the data transfer from and to host in serial fashion, while the data transfer between I/O and ring memory is in parallel fashion. Ring memory consists of as many planes as the number of layers involved-four, in this example-each of which is independently connected with I/O and processor array through the interconnection network. The processor array executes the basic intra-layer or inter-layer DRC instructions according to the command of the controller and updates the bit map data being circulated

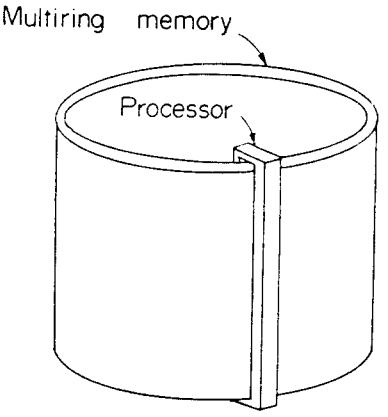


Fig. 3. An imaginary 1-Dimensional image prcessing scheme using ring memory.

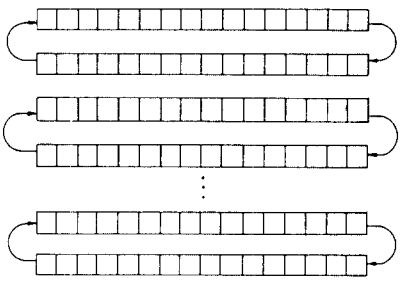


Fig. 4. Ring memory using shift register array.

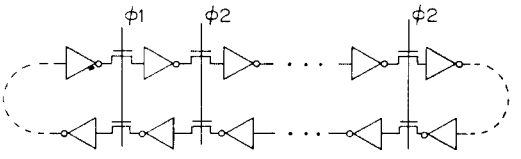


Fig. 5. Ring memory implemented with dynamic shift registers using 2-phase nonverlapping clocks.

in the ring memory. The instructions from the host are decoded by the instruction decoder into the precise control signals which then control the I/O and processor array one by one each cycle. Fig. 7 describes one slice of the MultiRing where the number of inputs or outputs is the number of mask layers, which is 4 in our example.

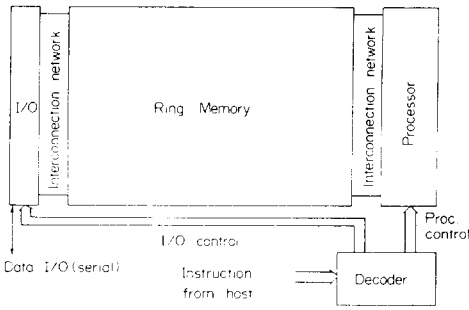


Fig. 6. Block diagram of multiRing. The processor part updates the bit map data in the ring memory.

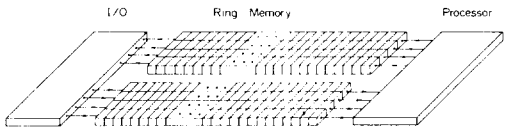


Fig. 7. One slice of MultiRing. The processor and I/O receive four bits from ring memory and return one of them updated while other three bits unchanged.

2. *Circuitry of unit processor*

Fig. 8 shows a block diagram of the unit processor in the linear processor array which consists of three switches (4x1-SW, 4x2-SW, 2x1 SW) and three submodules (detection-resizing module, Boolean module and output stage). 4x1-SW is a simple switch to select one layer out of the 4 input layers in the ring memory. We let C^t denote the selected pixel data in the current unit processor at time point t (t = -, 0, + signifies past, present and future, respectively)

The output of 4 x 1 switch will be stored in a 3 stage shift register as C^-, C^0, and C^+ constructing 3 x 3 window with those from the upper unit processor (U^-, U^0, U^+) and those from the lower unit processor (L^-, L^0, L^+). This 3 x 3 window becomes the data for the detection and resizing operations. On the other hand, C1, C2, C3, and C4 are also fed to 4x2-SW, where two (B1, B2) of them are selected as the inputs of the Boolean module which executes various Boolean operations such

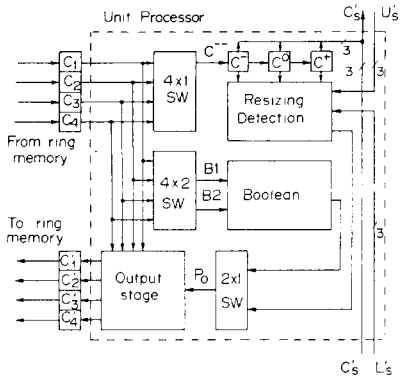


Fig. 8. Block diagram of the unit processor. number of input and output signals are four respectively. The signals (U-, UO, U+) and (L-, L0, L+) come from the upper and lower unit processors respectively to construct a 3 X 3 window with (C-, C0, C+).

as AND, OR, NOT and Copy between layers. The outputs of the detection-resizing and Boolean modules are fed to 2x1-SW whose output is Po. Po is fed to the output stage either to replace or to be ORed with one of C1, C2, C3, and C4. In the output stage, C1, C2, C3, and C4 are delayed by four clocks to be in the same phase with Po. The structures of the unit processors in the uppermost and lowermost rows must be designed to be slightly different from the inner ones since there is only one neighbor unit processor in these two boundary unit processors.

III. Definitions of basic instructions

The 16 basic instructions of MultiRing, which are summarized in Table 2, consists of 4 Boolean instructions, 8 resizing instructions, two detection instructions and two I/O instructions. When the bit value updated by the processor array is loaded back into the ring memory, two options are available. One is to write the updated value after clearing the previous value (write). Another option is to write the result of logical ORing between the new value and the previous one (paint). That is, mr_f(x,y,1) and mr_f(x,y,O) denote 'paint' and 'write' mode instructions, respectively, where mr_f( ) can be any MultiRing's basic

instruction with the input vector X and the destination register y, while the third argument, 1 or 0, determines whether to 'paint' or 'write'.

Table 2. MultiRing's basic instructions.

Instruction	number of arguments	Function description	Group
mr-AND	4	logical AND	Boolean
mr-OR	4	logical OR	
mr-NOT	3	logical negation	
mr-COPY	3	copy	
mr-SHR-NE	3	shrink to North-East	Geometrical
mr-SHR-NW	3	shrink to North-West	
mr-SHR-SE	3	shrink to South-East	
mr-SHR-SW	3	shrink to South-West	
mr-EXP-NE	3	expand to North-East	
mr-EXP-NW	3	expand to North-West	
mr-EXP-SE	3	expand to South-East	
mr-EXP-SW	3	expand to South-West	
mr-INDTC	3	interior detection	Detection
mr-EXDTC	3	exterior detection	
mr-IN	1	write on a layer	I/O
mr-OUT	1	read a layer to host	

1. Boolean instructions

The Boolean instructins that MultiRing can perform are AND, OR, NOT, and Copy between interlayers. Following examples show the usage of those instructions where A, B, C and D denote the layers.

```
mr OR(B,C,A,0);     /* Store A with OR
                       (B,C) */
mr AND(A,B,D,0);    /* Store D with
                       AND(A, B) */
mr NOT (A,C,0);     /* Store C with
                       NOT(A) */
mr COPY(B'A,0);     /* Store A with B */
mr OR (A,B,D,1);    /* Store D with
                       OR (OR(A,B), D)
                       or , paint D with
                       OR(A, B) */
```

2. Resizing (geometrical) instructions

There are two resizing instructins, 'Shrink' (SHR) and 'Expand' (EXP). When the 'Expand' instruction is executed for each geometrical primitive, its widths in X and Y directions are enlarged as much as 1 unit length. Although the ideal expansion or shrinkage is the enlargement or shrinkage of the original pattern by half unit length in +X, -X +Y, and -Y directions, we classified the EXP or SHR instructions according to the options as follows; (See Fig. 9)

```
Shrink = { SHR_NE, SHR_NW, SHR_SE,
           SHR SW }
Expand = { EXP_NE, EXP_NW, EXP_SE,
           EXP_SW. }
```

Examples:

```
mr_SHR_SE (A,D,O); /* After shrinking
                      layer A, move it
                      half unit length to
                      south-east direction
                      and put it on layer
                      D */
mr_EXP_NW(B,C,1); /* After expanding
                     layer B, move it half
                     unit length to north-
                     west direction and
                     paint it on layer C
                     */
```

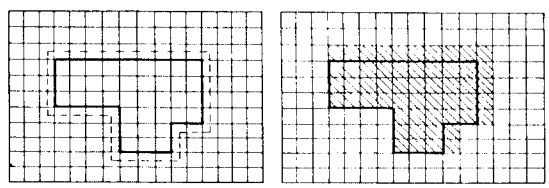


Fig. 9. (a) Ideal expansion in four directions by half grid length.
(b) Result of one of the four Multi-Ring expansion instructions, EXP-NE (Expand Northeast).

3. Detection instructions

Detection instructions are responsible for searching for the specific pattern in the given 3 X 3 window shown in Fig. 10, and consist of interior detection (INDTC) and exterior detection (EXDTC). These are described using

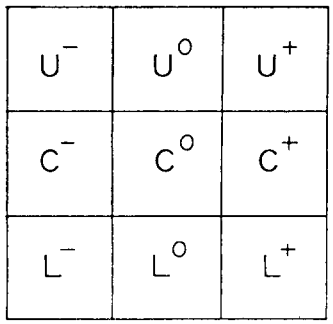


Fig.10. 3 × 3 window showing nine bit values available to each unit processor; U's, C's and L's are from the upper neighbor row, current row and lower neighbor row, respectively. Superscripts, -, 0 and + denote left neighbor column, current column and right neighbor column in the ring memory, respectively.

Boolean expressin as follows.

$$INDTC = C^0.((\tilde{U}^0.\tilde{L}^0) \dagger (\tilde{C}^-.\tilde{C}^+) + (L^-.U^+.\tilde{U}^-.\tilde{L}^+) + (U^-L^+.\tilde{L}^-.\tilde{U}^+)) \quad \text{eq.(1)}$$

$$EXDTC = \tilde{C}^0.((U^0.L) + (C^-.C^+) + (L^-.U^+.\tilde{U}^-\tilde{L}^+) + (U^-.L^+.\tilde{L}^-.\tilde{U}^+)) \quad \text{eq.(2)}$$

,where (tilde) denotes complementing.

Fig. 11. Shows four cases in INDTC which is used to check for the width rule in the 3 × 3 window. The precondition of INDTC is that C^0, the center of 3 × 3 window, should be 1, which is the first term of eq.(1). Fig. 11 (a) shows 2-λ width rule error in the vertical as well as horizontal direction which is reflected in the first and second terms within the outermost parenthesis (OR term) in eq.(1). Fig.11 (b) corresponds to the third OR-term in eq.(1) which checks for the width rule error in the diagonal direction. The fourth OR-term in eq.(1) is for checking in the other diagonal direction. No width rule error is reported for the pattern shown in Fig. 11 (c) since it represents a corner of a rectilinear polygon. The width rule error for the pattern shown in

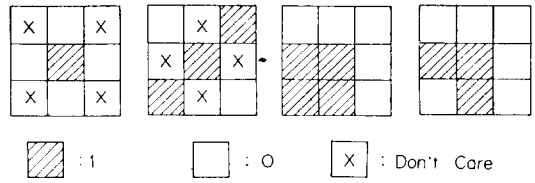


Fig.11. Several 3 × 3 window patterns for INDTC (Interior detection) operations.

Fig. 11 (d), error is not reported at the present time. Instead, it is to be reported when its neighbor cells, i.e., left neighbor cell (C^-) or lower neighbor cell (L^0) becomes the current cell (C^0). Exterior detection instruction, EXDTC can be explained in a similar fashion as INDTC except all the Boolean values are complemented.

Examples:

```
mr_EXDT(B,D,1); /* Paint the result of exterior
                   detection of layer B on layer
                   D. */
mr_INDTC(C,D,0); /* Write the result of in-
                    terior detdction of layer
                    C on layer D. */
```

4. I/O instructions

Data transmissions between MultiRing and host or display device (eg. CRT) are bit serial which reduces the bandwidth requirement of the channel and the number of external pins. mr_IN( ) instruction makes MultiRing receive bit data from host and load them on a specified layer in the ring memory. mr_OUT( ) makes MultiRing send bit data in the specified layer of the ring memory to host or display devices.

IV. Functional simulations of MultiRing

1. Programming for DRC applications

The design rules described in this paper were taken from [4]. Compared to any other hardware architectures, MultiRing is very flexible against the variations of design rules since the number of basic operations into which the required instruction is decomposed does not affect the hardware architecture.

The application programs running on Multi-Ring for DRC in IC layout were written in C language but presented in a simplified form here. The layers 2 and 3 in ring memory are reserved layers which are prohibited from being used as input layers.

2. *Width rule checking*

The following procedure, Width_rule_check (A,n) checks for the n-λ width rule in the mask layer A and reports the errors for all patterns having widths from 1 to (n-1) λ's. Since the interior detection operation, INDTC of Multi-Ring checks only for 2-λ width rule (where 1-λ width rule is illegal, while minimal 2-λ width is legal). Successive shrinking and interior detection instruction is required for reporting errors for all patterns having less than n-λ widths. The resultant errors are written on the layer 3 in the MultiRing memory.

```
Width-rule-check(A, n)
    /* Check n-lambda width spacing rule for layer A.*/
{
mr-INDTC(A,3,0);      /*Check2-lambda width rule*/
mr-COPY(A,2,0);       /*Copy from A to layer2.*/
for(i=1;i<= n-2;i++)
  {                   /*(n-2) repetitions of shrinking and*/
  mr-SHR-NE(2,2,0);/*interior detection detects(n-1)*/
  mr-INDTC(2,3,1); /*lambda width rule error.*/
  }
mr-OUT(3);            /*Output layer3 where errors are
                        stored*/
}
```

3. *Spacing rule checking*

Since the intra-layer spacing rule check is a problem which is complementary to the width rule check, the procedure, Space_rule_check_1 (A,n) can be understood in a very similar way. The inter-layer spacing rule check, Space_rule_check_2(A,n) needs extra processes such as zero inter-layer spacing check and the elimination of the bays in the two individual layers whose widths are smaller than n-λ's. The bay elimination prevents checking the intra-layer spacing errors in the two layers.

```
Space-rule-check-1(A,n)
   /* Check n-lambda intra-layer spacing rule for layer A*/
{
mr-EXDTC(A,3,0);   /* Check 2-lambda spacing rule for
                      A.*/
mr-COPY(A,2,0);    /*Copy from layer A to layer2. */
for(i=1;i<= n-2;i++)
  {                /*(n-2) repetitions of expansion
                      and*/
  mr-EXP-NE(2,2,0);/*exterior detection detects (n-1)*/
  mr-EXDTC(2,3,1);/*lambda spacing rule error.     */
  }
mr-OUT(3);         /*Output layer3 where errors are
                      stored*/
}

Space-rule-check-2(A, B,n)
                   /*Check n-lambda inter-layer
                     spacing*/
                   /*rule between layer A and B.  */
{
mr-EXP-NE(A,A,0); /*Checking zero inter-layer spacing
                     is*/
mr-EXP-NE(B,B,0); /*done by interior detection of the*/
mr-AND(A,B,2,0);  /* ANDed result of two expanded
                     layers.*/
mr-INDTC(2,3,0);
for(i=1;i<=n-1;i++) /* These expansion and shrinkage
                       */
  {                 /* fill bays in layer A and B*/
  mr-EXP-NE(A,A,0); /*whose widths are less than n*/
  mr-EXP-NE(B,B,0); /*in order to prevent reporting*/
  }                 /*intra-layer spacing errors in*/
for(i=1;i<=n;i++)   /*layer A or B.              */
  }
  mr-SHR-SW(A,A,0);
  mr-SHR-SW(B,B,0);
  }
mr-OR(A,B,2,0);     /*Inter-layer space error
                      between*/
Space-rule-check-1(2,n);/*A and B is equivalent to
                          intra-  */
{                   /*layer error in layer2 which is
                      */
                    /*ORed result of layer A and B.
                      */
```

4. *Externsion and Enclosure rule checking*

The extension rule check is to report for insufficient amount of extension of the patterns in one layer over the patterns in another layer, for example, polysilicon over diffusion, depletion implant over polysilicon, etc. The procedure Extension rule check(A,B,n) consists

of 1-λ expansion of all the patterns in layer B in four directions and subtraction of layer A from layer B, which is followed by (n + 1) λ width rule check for layer B. The extension rule check algorithm can be used to check for the enclosure rule error, where layer A is regarded as the contact cut layer.

```
Extension rule check (A,B,n)

/* Check for the extension of patterns in layer
   B over the / *
   boundary of patterns in layer A by at least n
   lambda's */

{

mr EXP NE(B,B,0);   /* Grow all patterns
                       in layer B in four   */
mr EXP SW(B,B,0);   /* directions.          */
mr INTV(A,A,0);     /* Eliminate A from B
                       by ANDing B with */
mr AND (B,A,2,0);   /* the complement of A.*/
Width rule check 2,n+1);
}
```

5. *Simulation and its results*

A software simulator for MultiRing was written in C language, whose environment is schematically shown in Fig. 12. Each procedure within this software simulator has a one-to-one correspondence to each hardware module within the MultiRing. The software library for MultiRing's basic instructions consists of "I/O" and "Processor" procedures which correspond to I/O and processor blocks respectively, while "ring memory" procedure within this software simulator is an integer array performing the function of MultiRing's ring memory. A graphics terminal is controlled by "I/O" procedures to dump the contents of ring memory onto the screen. The application program calls the functions in the library of MultiRing's basic instructions to update the contents of ring memory array. This application program can be directly applied to the actual MultiRing hardware.

Simulation results of DRC for an example shown in Fig. 13(a) is shown in Fig. 13(b). This example consists of rectilinear patterns in layer A and B. Four kinds of design rules,

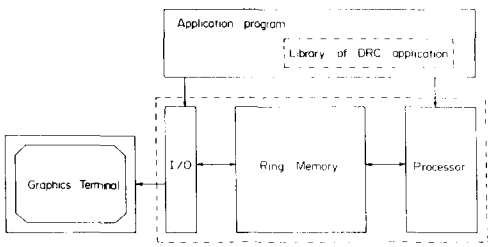
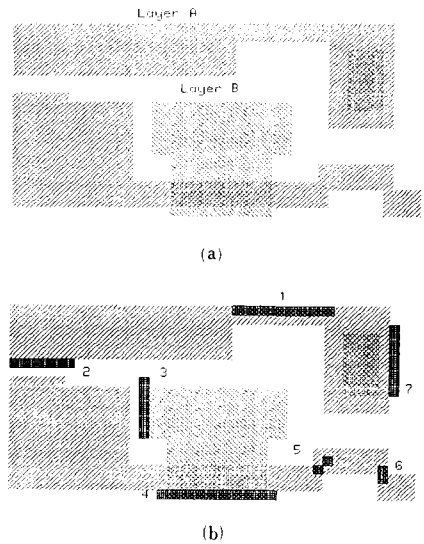


Fig.12. Block diagram of software simulator for MultiRing. The portion within the dotted rectangle corresponds to the MultiRing hardware.



1 ;3-λ width error for layer A.
2 ;3-λ spacing error for layer A.
3 ;3-λ spacing error between layer A and B.
4 ;2-λ extension error of layer B from A.
5,6;3-λ width error to diagonal direction of layer A.
7 ;2-λ enclosure error of layer A from B.

Fig.13. Simulation result.
(a) Two input layers A and B.
(b) Simulation results of MultiRing for the given input layers. Location of various DRC errors are shown with their identification numbers explained below.

that is, width, spacing, extension, and enclosure rules were tested successfully. In Fig. 13(b), "1" reports 3-λ width error for layer A. "2" reports 3-λ spacing error for layer A. "3" reports 3-λ spacing error between layer A and B. "4" reports 2-λ extension error of layer B over A. "5" and "6" reports 3-λ width error to diagonal direction of layer A. "7" reports 2-λ enclosure error of layer A for B.

V. Conclusion

An efficient hardware architecture called MultiRing suitable for various 2-dimensional rectilinear geometry processing is proposed. Although its application has been discussed only for design rule checking in VLSI layout, MultiRing can easily handle various image processing operations such as filtering, translation, correlation and contour extraction. While MultiRing is basically a hardware tradeoff between single pipelined processor and 2-D processor arrays, its most salient feature is the flexibility in terms of design rule changes or instruction set enhancements. Finally, correct functional behaviour of MultiRing has been shown by a software simulator (Written in C) having a one-to-one structural correspondence with the proposed hardware.

References

[1] Tom Blank, Mark Stefik and Willem van-Cleemput, "A parallel Bit Map processor architecture for DA algorithms". *Proc. 18th Design Automation Conference,* pp. 837-845, 1981.

[2] Ravi Nair, Se June Hong, Sandy Liles, and Ray Villani, "Global Wiring on a Wire Routing Machine", *Proc. 19th Design Automation Conference,* pp. 224-231,

[3] Rob A. Rutenbar, Trevor N. Mudge and Daniel E. Atkins, "A class of cellular architectures to support physical design automation", *IEEE Transaction on CAD of IC's and systems,* vol. CAD-3, no. 4, pp 264-278, October 1984.

[4] Carver Mead and Lynn Conway, *"Introduction to VLSI system",* Addison Wesley, 1980, Chapter 2.