A Parallel Bit Map Processor Architecture for DA Algorithms

by Tom Blank, Mark Stefik, and Willem vanCleemput

Stanford University and Xerox PARC
Stanford, California and Palo Alto, California

1. Abstract

Bit maps have been used in many Design Automation (DA) algorithms such as printed circuit board (PCB) layout and integrated circuit (IC) design rule checking (DRC). The attraction of bit maps is that they provide a direct representation of two-dimensional images. The difficulty with large scale use of bit maps (e.g., for DRC on VLSI) is that the large amounts of data can consume impractical amounts of computation on sequential machines.

This paper describes a processing architecture that is specifically designed to operate on bit maps. It has an inherently two-dimensional construction and has a very large parallel processing capability.

Also included in this paper are descriptions of algorithms that exploit the architecture. Algorithms for routing, DRC, and bit vector manipulation are included.

2. Introduction

As the size of design problems increases, the computational time required by the software tools also increases. Unfortunately, many DA programs have a non-linear run time dependence on the size of the design problem. Current size problems often require run times of several hours. For example, PCB routing for a 10 x 10 inch board with 100 IC's and DRC on a 200 x 200 mil integrated circuit require several hours of CPU time on a medium-sized processor.

There are a number of possible solutions to the size vs. execution time problem. The most obvious solution is to use faster and larger computers with the same programs. This approach is doomed to failure for two reasons: very large cost of large systems; the problem size may increase beyond the capability of the machine. Unfortunately, the speed of large mainframes is not increasing as fast as the increase in problem size and commensurate non-linear increase in execution time. Another possible solution is to build special purpose hardware that physically implements an algorithm. The Lee routing algorithm is an example that has been implemented with dedicated hardware. However, this approach also has weaknesses: no flexibility to adapt to changing technology, large expense due to low volume of the device and limited useful lifetime.

Reasonable solutions to the size vs. execution problem are: develop better algorithms (i.e. algorithms that have polynomial degree closer to one); define better design methodologies (again giving better algorithms); identify heavily used (i.e. bottleneck) operations for many DA algorithms and develop hardware to implement the general operations. The hardware developed for multiple purposes is very different from special purpose hardware (such as a Lee router box) and its life expectancy is longer. The longer life is attributed to its inherent flexibility which can adapt to changing technologies. Additionally, the cost of the device can be amortized over many applications.

A bit map has been identified as a commonly used data structure for representing physical structures (i.e. PCB or IC masks). Additionally, bit map operations have also been identified as a bottleneck due to: large amounts of data; a miss match between conventional word based computers and required bit operations on two dimensional data structures.

It should be noted that bit map processing architectures have been proposed since 1958 [9] for image processing applications. Recently, machines have been reported by Reeves [6, 7] and Duff [2, 3]. However, in all the earlier work, a large scale implementation was problematic.

We describe a physical chip design that can be integrated into a large scale bit processing architecture. The approach is to incorporate one processing element into each node of a bit map. The scheme is called SAM for synchronous active memory. In the following sections, the operation of one SAM cell, the overall SAM architecture, algorithms and future work are described.

3. A Bit processing element

A SAM cell is composed of seven major blocks connected as shown in figure 3-1. Each cell can be viewed as a completely independent one bit processor with the following attributes:

- 16 one bit registers; These can be viewed as layers of sixteen different bit maps.
- 20 (assembly level) instructions
- Single operand instructions with the result stored into the one bit accumulator
- Direct connection with nearest four neighbors
- Connection with global OR output line
- Connection with global input line
- A cell must be externally enabled by both a row and column select line for an instruction to be executed.

The instruction set for one SAM cell is broken into four categories: boolean, load/store, read/write and neighbor instructions. Each is discussed separately in the following sections. A complete instruction set is given in figure 3-2.

3.1. Boolean Instructions

All boolean functions of two bits are implemented. The accumulator always supplies one operand with the other operand supplied from one of the 16 internal registers, the result is always stored in the accumulator. For single operand instructions, the accumulator is both the source and destination. For all boolean operations, the result of the operation is also output to the global wired OR external output line. This feature is useful for doing an operation over an area in the bit map and testing the result.

0146-7123/81/0000-0837$00.75 © 1981 IEEE

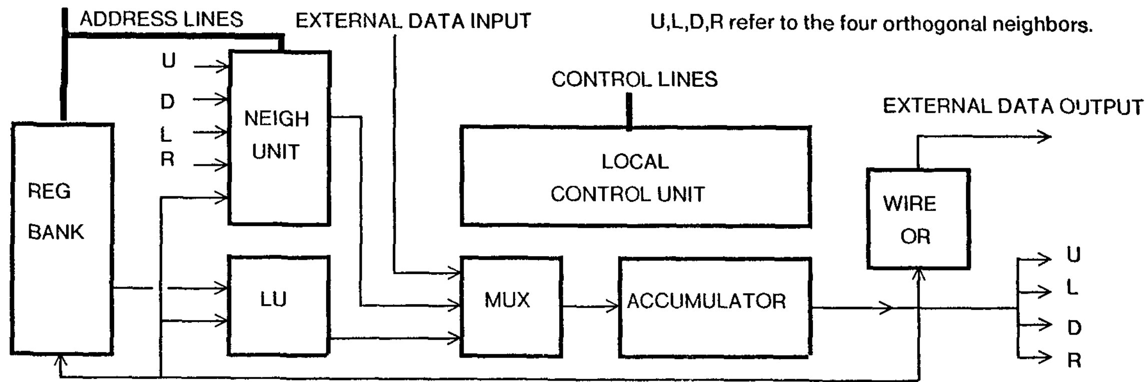

Figure 3-1: SAM Cell Block Diagram

BINARY OP CODE		OPERATION
		Read/Write Instructions
00011	READ --	ExtDataOut := Acc (ORed over region)
10100	WRITE --	Acc := ExtDataIn
		Load/Store Instructions
10010	STORE<bit>	<bit> := Acc
00101	LOAD<bit>	Acc := <bit>
		Neighbor Instruction
10001	NEIB<bit>	Acc := $M_0Nbr_U + M_1Nbr_L + M_2Nbr_D + M_3Nbr_R + M_4Nbr_C$
		(note: M_0-M_4 refer to the mask bits. See Fig. 3-4)
		Boolean Instructions
00000	CLR --	Acc := 0
00001	AND<bit>	Acc := Acc and <bit>
00010	GT<bit>	Acc := Acc greater-than <bit>
00100	LT<bit>	Acc := Acc less-than <bit>
00110	XOR<bit>	Acc := Acc xor <bit>
00111	OR<bit>	Acc := Acc or <bit>
01000	NOR<bit>	Acc := Acc nor <bit>
01001	CMP<bit>	Acc := Acc equals <bit>
01010	NOT<bit>	Acc := not <bit>
01011	LE<bit>	Acc := Acc less-than-or-equals <bit>
01100	INV --	Acc := not Acc
01101	GE<bit>	Acc := Acc greater-than-or-equals <bit>
01110	NAND<bit>	Acc := Acc nand <bit>
01111	SET --<bit>	Acc := 1

Figure 3-2: SAM Instruction Set

3.2. Load/Store Instructions

These instructions move data between the accumulator and one of the cell's internal registers. The instructions are: move data from the accumulator to an internal register (Store) and move data from an internal register to the accumulator (Load). Store is the only instruction that modifies the value of a register.

3.3. Read/Write Instructions

These instructions move data between the accumulator of selected SAM cells and the external SAM controller. In the context of only one cell, Read takes data from the external data input line and puts it into the accumulator. Write places the contents of the accumulator on the external data output line. This permits an external device to exchange data with a single cell. When more than one cell is enabled, the Write instruction actually gives the wired OR operation over the values of the selected accumulators as the output.

3.4. Neighbor Instruction

The neighbor instruction provides the mechanism to exchange data from one SAM cell to its four orthogonal neighbors (See figure 3-3). The data exchange is only between the accumulators. A five bit mask is used to select which of the cells will be used. The data from the selected cells are combined using an OR operation and the result is stored in the accumulator. This operation is done simultaneously for all selected cells.

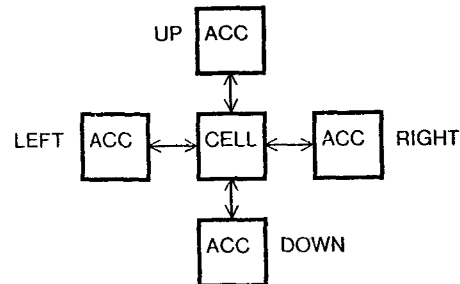

Figure 3-3: SAM Neighbor Instruction

The neighbor instruction is the most powerful instruction of a SAM cell since it uses the inherent two dimensional configuration of the bit map. With this instruction, it is possible to transfer blocks of data (by enabling a region), up, down, right and left by a simple mask specification (see figure 3-4). The neighbor instruction is powerful for data movement since the required execution time depends only on the distance moved not the amount of data (assuming the bit map has been previously loaded). Additionally, more complicated operations, which combine data from different neighbors, can be done by selecting a mask with more than one neighbor selected. If all the mask bits are set, data propagates isotropically.

3.5. Physical Implementation

The practical realization of a SAM depends on the development of a design for a single processor that is sufficiently small that it can be

EXAMPLES :	U	L	D	R	C
Transfer UP	1	0	0	0	0
Transfer LEFT	0	1	0	0	0
Transfer DOWN	0	0	1	0	0
Transfer RIGHT	0	0	0	1	0
Horizontal Transfer	0	1	0	1	1
Vertical Transfer	1	0	1	0	1
Isotropic Transfer	1	1	1	1	1

Mask Bits

NEIGHBOR
U - Upper
L - Left
D - Down
R - Right
C - Cell

Figure 3-4: Masks for Neighbor Instruction

replicated enough times to contain (or reasonably partition) a non-trivial sized problem. A prototype has been designed using an NMOS process [5] and is currently in the fabrication process.

For the first SAM cell prototype, the masks for only one cell were designed. The cell size is approximately 900 x 900 microns (this does not include external connection pads). This size is based on a fabrication process which had a lambda of 2.5 microns (lambda is one half the minimum feature size of the process). The design contained approximately 350 transistors in 165 logic gates (pass transistors were counted as a gate). The SAM cell mask design was done so that a step and repeat operation could be used to fabricate a larger array after the design is verified.

Many architectural decisions and design observations were made that had a direct influence on the current size of the design. Future prototypes will be able to use both the hardware and architectural information from the first chip. The following observations had a major impact on the prototype design:

- Each SAM cell is greatly simplified by *not* having a stored program. All sequencing of operations is centrally controlled by the master which broadcasts instructions to the SAM array. Additionally, the master selects the regions in the array that are enabled to execute an instruction. The adoption of global control eliminates the need for complicated control logic and additional storage for instructions in the individual cells.

- Single operand instructions were adopted due to the commensurate simplicity of the data paths and logical operations.

- A fully static register bank (memory) was used. This permits easy testing of the prototype design. This choice had the largest impact on the size of the implementation since the sixteen register cells occupy over 60% of the layout.

- All boolean operations of two variables are easily implemented with a four to one multiplexor by choosing the instruction codes properly. Additionally, no arithmetic instructions were implemented since all operations can broken into their boolean components. This at first inspection seems woefully inefficient; but, the vast parallelism of a SAM makes the operations on an entire bit map orders of magnitude faster than conventional sequential machines.

- For a large scale implementation, it is important to get as many SAM cells on each chip so that only a limited number of chips are needed. Since the number of external chip connections is primarily a function of the number rows and columns on the chip, the minimal number of external connections was a design goal. Therefore, bi-directional data lines were incorporated between all neighbors. It increased the complexity of the internal circuitry in each node for a significant decrease in the pin out (See figure 3-5).

- The required AND/OR logic of the neighbor instruction was easily implemented with one NMOS gate.

- The address lines used to select a register for a boolean operation are not used during a neighbor instruction. This permited a great savings in required lines by using the unused address lines as the neighbor instruction mask bits.

Signal	Number of Pins
Bit Addresses	5/chip
Operation Codes	5/chip
Power/Ground	2/chip
Clock	1/chip
Communications Lines	2/chip
Row Select	1/row
Column Select	1/column
Neighbor Connections	2/row + 2/column

Figure 3-5: SAM Pin Connections

An important aspect of the SAM cell is the maximum speed of operation. For NMOS, a performance measure is the propagation delay time of two inverters in series, called the 'pair delay'. The NMOS process used for the prototype had a pair delay of approximately 8 nanoseconds. For a SAM cell, the speed limiting signal paths can be separated into two distinct groups: internal delay and external delay. The maximum internal delay is during boolean instructions where an operand fetch from the register set is made. The predicted delay is 100 nanoseconds. External delay is when one SAM cell is communicating outside of its cell boundaries. For all instructions (when the cell is enabled) the accumulator value is transmitted to the wired OR data line. External time delay depends on the following factors: capacitance of the external data line, impedance of the cell pull down transistor and physical construction of the logical OR function through out the SAM array.

4. Bit Map Processing Architecture

There are two SAM architectures considered: a fully implemented SAM array where each bit map node is one SAM cell and a virtual SAM array where the logical bit space is larger than the physical implementation. Each is discussed separately.

4.1. Full Hardware SAM array

A complete hardware SAM implementation (one SAM cell per bit map node) is very attractive due to its tremendous parallel processing capability and speed (See figure 4-1). Additionally, the SAM array control and data requirements are simple. All data transfers between the Bit Map Processor and the conventional computer system are through either the one bit external data I/O line or from the computer data bus into an edge register. The Bit Map Processor control simply regulates the data exchange and coordinates SAM array instruction execution. All SAM instruction sequencing is done by a conventional computer system.

There are a number of tradeoffs and problems with a fully implemented Bit Map Processor. Physical realizability is one of the obvious problems. A SAM array 1K by 1K would require over 4096 individual chips if each SAM array chip contained an array of 16 x 16 processors. A 16 x 16 SAM array chip can be built if one SAM cell can be built in a 400 x 400 micron square and a .6 x .6 cm chip is used. The

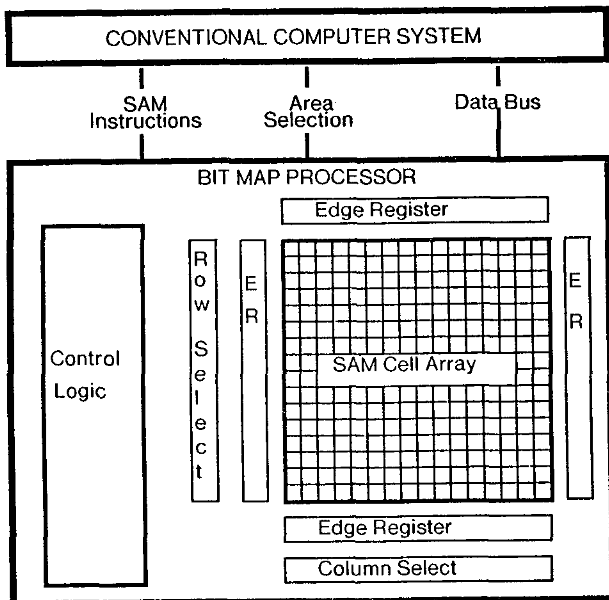

Figure 4-1: Bit Processing Architecture using one SAM Cell per Node

prototype SAM cell (900 x 900 microns) can be reduced if a smaller geometry process is used and dynamic instead of static registers are used in each cell. Another difficulty of a 16 x 16 chip is that 111 pins are required. The instruction cycle time is a function of internal cell, external cell, external chip and external board interconnections. To accommodate the longest delay path, the calculated instruction cycle time is 300 nanoseconds (using a hierarchical OR gate connection). Other architectural constraints are: a fixed bit map size (i.e. 1K x 1K in the example), a fixed shape (i.e. square or rectangular) and a fixed number of registers in each cell (i.e. sixteen for the current SAM cell).

4.2. Virtual SAM Architecture

Another possible solution to the implementation problem is to relax the requirement that each bit map node have a dedicated processor. This allows one processing node to process many nodes of a bit map. The complete architecture of a 1K x 1K virtual bit map processor is shown in figure 4-2. The main component is a 32 x 32 array of SAM cells that have been modified as shown in figure 4-3. There are two changes from the previously described SAM cell. The accumulator and register bank are removed and replaced by a 1K and a 16K RAM, respectively. The external RAM attached to a SAM cell defines the allowable configurations of the virtual bit map; For example, using the memory components specified, the largest virtual bit map is 1K x 1K with 16 registers in each cell (this is identical to the full hardware implementation). The minimum size map is 32 x 32 with 16K registers. (Note: the maximum bit map size is limited by the memory attached to the accumulator. If a 16K RAM is used for the accumulator set, a 4K x 4K virtual map with one register bit per cell may be configured). The only restriction on the allowed bit map shapes is that the smallest segment is 32 x 32 and only rectangles are allowed.

The control of the virtual Bit Map Processor is composed of seven primary components: Program control logic, Primary address control, Edge 1 - 4 control and the SAM cell array. The controllers handle the sequencing of a SAM instruction over the virtual map segments on the physical 32 x 32 processors. For simple accumulator and register SAM instructions (i.e. booleans) the controllers broadcast the instruction and the address of the selected register and accumulator. The same instruction is then repeated over the different bit map segments until the entire virtual bit map has been processed. The primary address controller supplies the register and accumulator addresses based on the selected register in the instruction and the currently active segment.

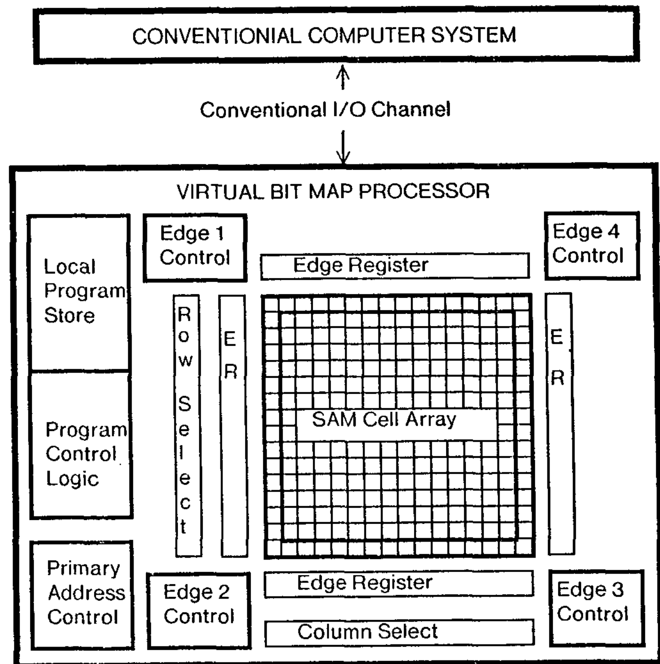

Figure 4-2: Virtual Bit Map Architecture

Neighbor instructions are much more difficult since the values of the four orthogonal neighbors have to be supplied. For internally located cells, all neighbor values are contained in the current segment; however, cells located on the edge of the physical array require neighbor values that are contained in other segments. The problem is solved by splitting the edge cell accumulator set into two banks (three banks for the four corner cells). Each bank can be controlled by either the main address controller or the by the edge address controller. Multiple accumulator banks permit simultaneous access of different segments. The final complication for the neighbor instruction is that the memory partitions must be folded so that the required neighbor values are located in the same accumulator set (See figure 4-4). Folding the bit map requires that the neighbor mask be modified depending on the currently active segment.

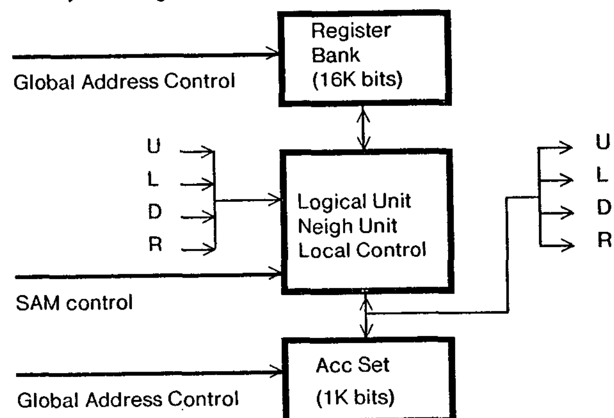

Figure 4-3: Modified SAM Cell

Another part of the virtual architecture is a local program store. Since the execution time of an instruction depends on a number of variables, it would be impractical to burden the conventional computer system with instruction sequencing. A SAM array program is simply downloaded from the conventional computer memory. The Bit Map Processor then signals the host when the program segment execution is complete.

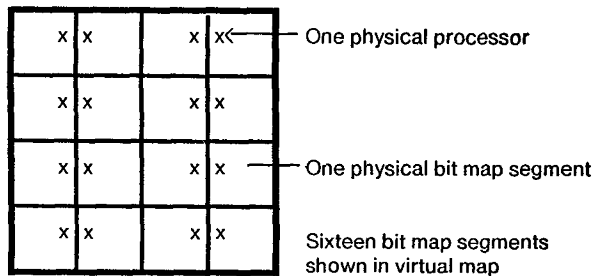

Figure 4-4: Memory Partition

The virtual bit map scheme has a number of important advantages over the complete hardware SAM array implementation:

- Easier physical implementation since fewer chips
- Only $K*(N/m)^2$ slower than a complete hardware SAM implementation; N x N is the virtual array size; m x m is the hardware segment size; K is a constant less than one that is a function of the external cell delay due to the external OR function output (Note: ideally $K \propto 1/\log(N/m)^2$).
- Easy reconfiguration of bit map size, shape and number of registers
- Utilization of standard RAMs for the bulk storage

The chip count for a 1K x 1K virtual array is approximately 2K chips if one chip is used for each accumulator set, register bank and for 32 SAM cells on one chip; however, both accumulator and register memory can be shared between multiple SAM cells. A 16K by 4 bit chip can be used for four register banks. A 1K by 8 bit chip can be used for eight accumulator sets. Both changes reduce the chip count below 500. The instruction cycle time is primarily determined by the memory cycle time since the physical SAM array and external delays are short. A faster cycle time can also be achieved using a cache memory scheme for both the accumulator set and register bank. A minor cycle time (only for the 32 x 32 segment) of 100 nanoseconds is attainable since the array size is physically small (minimal external delay) and if the internal delay is reduced using the memory caching technique. The cycle time of an N x N virtual array (using the 32 x 32 array) is 100ns * $(N/32)^2$.

5. Algorithms for a Parallel Bit Map Architecture

Many algorithms currently used in design automation are inherently parallel. The two most obvious examples are the Lee Maze router [4], and the Soukup Global router [8]. Other examples are design rule checking, component placement and circuit extraction. Bit vector manipulations take advantage of both the parallelism and bit handling nature of an active bit map.

In the following sections, parts of algorithms will be presented that demonstrate both the flexibility and power of a SAM array.

5.1. Simple Boolean operation

Figure 5-1, shows the simple boolean operation AND between two planes of a bit map. In the example, all the bit map cells are enabled. The external output data line (the wired OR of all enabled cells) is set to a one after the statement execution. This is only one instruction.

5.2. Simple Neighbor operation

Figure 5-2, shows a simple neighbor operation when all the mask bits are set to 1 (Isotropic transfer). The logical result is that each cell takes the logical OR of itself and its four orthogonal neighbors. All bit map cells are enabled. The physical result is that each object in the map is expanded by one cell (note: a four corner expand excludes all diagonal neighbors). Again, this is only one instruction.

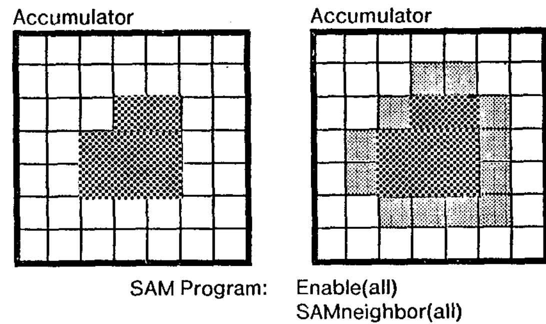

Figure 5-2: Four Corner Expand

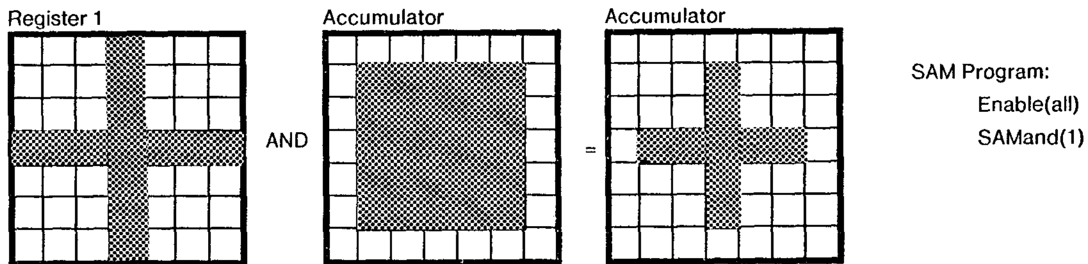

Figure 5-1: Boolean Example

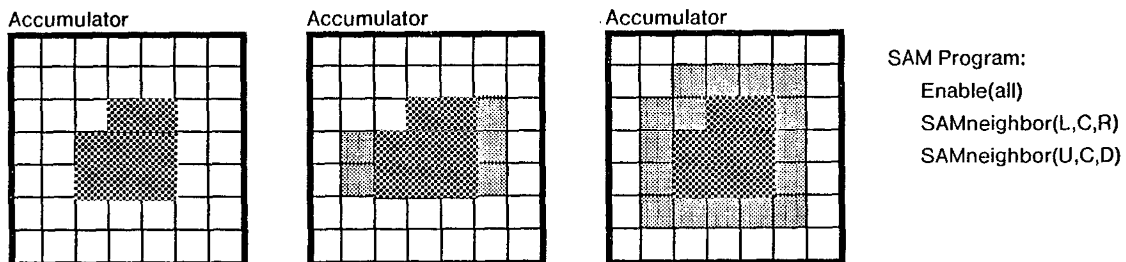

Figure 5-3: Eight Corner Expand

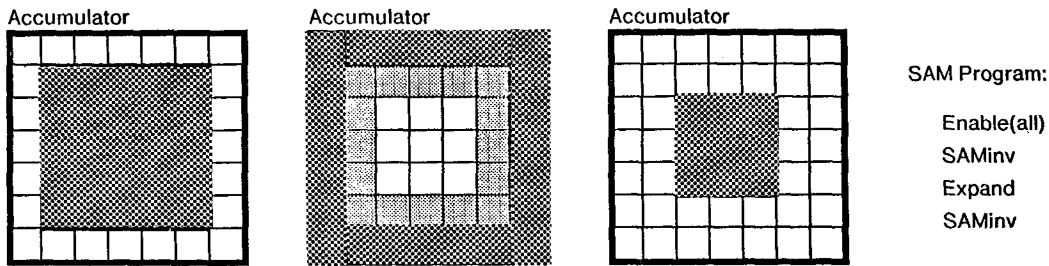

Figure 5-4: Shrink

An expand that includes diagonals (an eight corner expand) can be done in two instructions: one horizontal transfer and one vertical transfer. A horizontal transfer puts the logical OR of itself and two horizontal neighbors into the accumulator. A vertical transfer puts the logical OR of itself (which now contains the OR of itself and two horizontal neighbors) and two vertical neighbors into its accumulator. The logical result is the logical OR of itself and eight nearest neighbors (See figure 5-3).

5.3. Simple Instruction Sequence: 'Shrink'

Shrink, the inverse function of expand, is done in three or four instructions: complement, expand (either four or eight corner expand) and complement (See figure 5-4).

5.4. Lee Router

A conventional Lee router is composed of three phases: wavefront expansion, trace back, and map clearing. A typical Lee wave expansion is shown in figure 5-5 for a single layer where there are four values allowed in each bit map node: available (space), occupied (X), wavefront1 (1), and wavefront2 (2) [1]. The symbols 'S' and 'D' represent the source and destination of the wave front and are not actually stored in the bit map. The return path (in the example) is found by following a 2112 sequence.

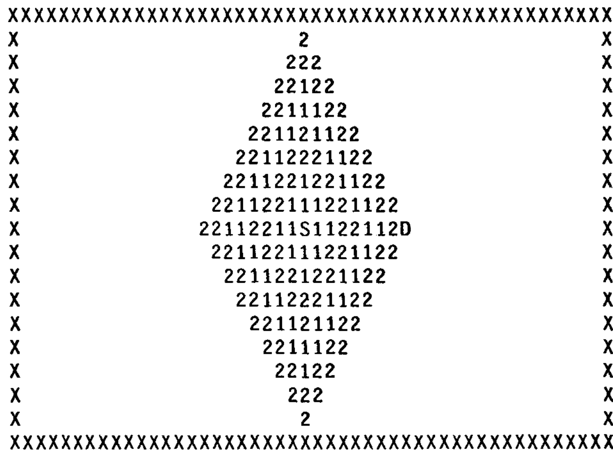

Figure 5-5: Conventional Lee Router

A program for a SAM array is given that implements the conventional Lee router. As in figure 5-5, there are four allowed values; but are represented using three bit planes. Register zero contains only a map of the obstructions. Register one contains the wavefront1 expansion and register two contains the wavefront2 expansion. All unoccupied bits are assumed available. The Lee router program is given in a Pascal like notation where each SAM instruction is specifically marked with the prefix SAM. The enable procedure selects the area within the SAM array to execute the broadcast instructions. Figure . shows the wave expansion steps of a simple routing. The 'x's repres occupied bits in the map.

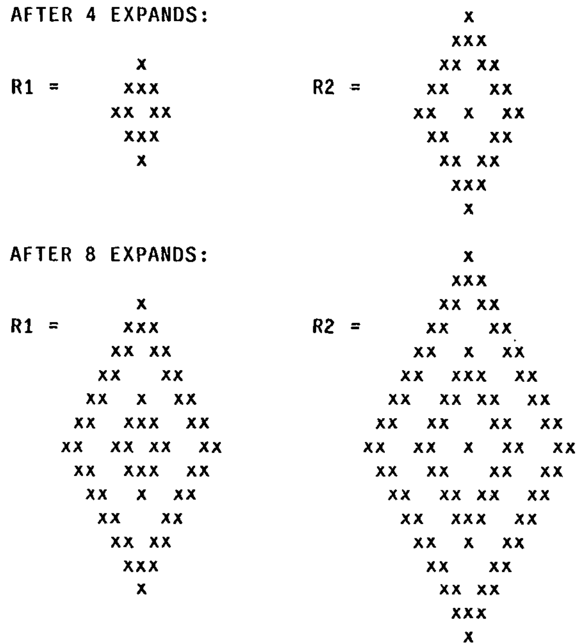

Figure 5-6: SAM Implementation of Lee wavefront expansion

Using the SAM instruction cycle times of 300ns (refer to section 4.1) for the 1K by 1K hardware implementation and 100ns per minor cycle (refer to section 4.2) for the virtual bit map machine, the estimated execution time of the Lee algorithm can be calculated. The examp problem is for a printed circuit board routing where a 512 x 512 bit m. is required. There are 1000 traces with an average length of 200 gri The execution times for the SAM architectures are calculated computing the time for one 200 grid route and multiplying by the t number of traces. The constant mapinit (1-5 seconds) is the t required to initially load the bit map pads and obstructions be routing. The timing of a conventional computer is also shown comparison.

SAM Architectures	Execution Time
1K x 1K hardware map	.4 sec. + mapinit
32 x 32 virtual array	15 sec. + mapinit
Conventional Computer	5 Hours

Figure 5-7: Lee Routing Timing

```
{Lee Maze Router Main Program

**** SAM register uses ****

    R0 - Obstructions
    R1 - Ones layer
    R2 - Twos layer

xstart,ystart - starting location
xstop,ystop   - destination for routing }

Begin (* MAIN *)

{** map initialize **}
SAMenable(all);
SAM(clr);          { clear layer }
SAMenable(xstart,ystart,xstart,ystart);
SAM(sett);         { enter starting seed }
SAM(store,0);
SAMenable(xmin,ymin,xmax,ymax);
SAM(store,2);      { save entire layer }

{** wave expansion **}
cnt := 0;
repeat
  step := cnt mod 4;
  SAM(neib,all);
  SAM(gt,0);
  if step = 1 then
    begin SAM(gt,2); SAM(store,1) end;
  if step = 3 then
    begin SAM(gt,1); SAM(store,2) end;
  cnt := cnt + 1;
until hittarget(xstop,ystop);

{**trace back **}
x1 := xstop; y1 := ystop; dir := 0;
SAMenable(x1,y1,x1,y1);
SAM(store,0);      { put in endpoint }
for targetlayer := cnt-2 downto 0 do
  begin
  step := targetlayer mod 4;
  chasemarkfrom(step,dir,x1,y1);
  end;
End. (* MAIN *)
```

```
function hittarget(x,y:integer):boolean;
{ true if the destination has been marked }

begin
SAMenable(x,y,x,y);
SAM(readd);
hittarget := SAMflag;
SAMenable(xmin,ymin,xmax,ymax);
end;

procedure chasemarkfrom(trgtlayer:integer;
                        var dir,x,y:integer);
{ finds the best path from the specified
  point. It first tries the same direction
  then looks in the others                  }

Label 1;

Var
 trydir:          integer;
 newdir:          integer;
 xrel,yrel:       integer;

Begin
For trydir := 0 to 3 do
    begin
    newdir := (trydir+dir)mod 4;
    case newdir of
       0:  begin xrel := 0; yrel := 1 end;
       1:  begin xrel :=-1; yrel := 0 end;
       2:  begin xrel := 0; yrel :=-1 end;
       3:  begin xrel := 1; yrel := 0 end
       end; {end case}
    SAMenable(x+xrel,y+yrel,x+xrel,y+yrel);
    if trgtlayer < 2 then SAM(load,1)
      else SAM(load,2);
    if SAMflag then goto 1;  {exit if found}
    end; {end of for loop}

1: { new location found }
dir := newdir;
x := x + xrel;
y := y + yrel;
SAM(store,0);      { enter new point }
End;  { end of chasebackmark }
```

5.5. Comments on Soukup's Global Router

Soukup [8] presented a new philosophical approach to routing where all the traces are routed simultaneously (called GR1). Additionally, another algorithm was presented to connect problem traces (GR2) again using an approach where all traces are effected simultaneously. We will not present a SAM implementation but a number of comments and observations are appropriate.

It is important to note that both global algorithms GR1 and GR2 are identical with the exception of priority assignment. The primitive operations of both are:

- Checking for local bottlenecks.
- Verifying that all local bottlenecks are true bottlenecks. The standard Lee router can be used for this check. 'Dead space' is also reset during this phase.
- Expansion of the wave front according to priorities and physical conditions.
- Assign priorities to all cells according to algorithm rules.
- Reduce wave front to a standard line route.

With the exception of priority assignment and sequencing between the primitive operations, all the required operations can be easily implemented on a SAM array. This is due to the locality of information near each cell.

5.6. IC Design Rule Checking

Integrated circuit design rule checking is another type of algorithm that is well suited to a SAM implementation. Many of the typical geometrical operations used in DRC like: expand, shrink, merging of IC layers and intersection are single SAM instructions or simple sequences of SAM instructions. Both expand and shrink have been mentioned in earlier sections. Merging is a simple bit map boolean OR between layers and intersection is an AND operation with area check.

A very simple check is presented in figure 5-8 for the metal coverage of cuts. The design rule in the example requires that all contact cuts are covered by one grid unit of metal in all directions. The test is composed of three SAM instructions: expand (8 corner) all contact cuts and XOR with metal layer. Any design rule violations will have a non-zero accumulator.

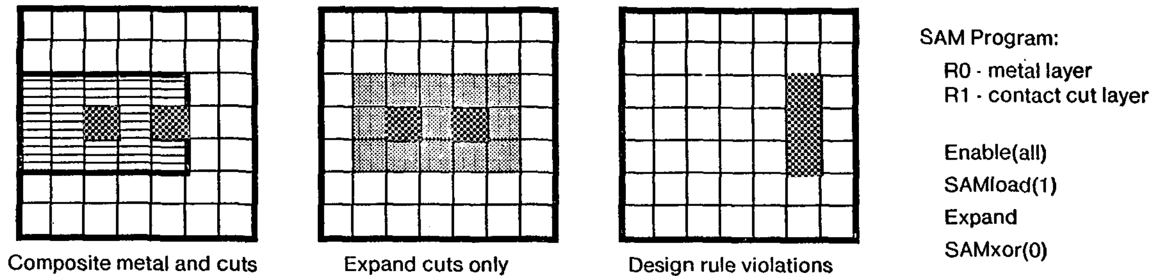

Figure 5-8: Design Rule Checking: Contact Cut Spacing

The use of a SAM does not increase the value of a geometrical DRC. All the problems of a bit map approach are still present: only spacing type checks, diagonal line problems and large bit map size; however, SAM does decrease the execution time. With SAM, the time dependence of many algorithms is completely data independent. For example, the contact coverage check only requires three SAM instructions (The initial time required to fill the bit map with the mask information is ignored.).

5.7. Other DA algorithms

5.7.1. Bit Vector operations

There are a number of problems with the implementation of bit vector operations on conventional word based computers. The two major problems are: length of bit vectors (requiring many computer words) and the mapping of standard bit vector operations into conventional computer operations. Standard bit vector operations include:

- complement
- intersection: A ∩ B
- union: A ∪ B
- cover: A intersect B = A; means B covers A
- sharp: A intersect (complement (B))

The use of SAM, inherently designed to manipulate binary bits, would solve the standard computer word length problem. The mapping of bit vector operations into SAM instructions is not a perfect match but the inherent gains of parallelism completely over rides the difficulties. Figure 5-9 is an example of the intersection operation.

Allowed Bit Vector Values

```
0 - binary zero
1 - binary one
X - don't care
```

Internal SAM Representation

```
10 - binary zero
01 - binary one
11 - don't care
00 - illegal value
```

Vector example 1

```
XX0 ∩ X10  =  X10
(11 11 10) AND (11 01 10) = 11 01 10
```

Vector example 2

```
X00 ∩ 110  =  Ø
(11 10 10) AND (01 01 10) = 01 00 10
      (the 00 bit denotes a null intersection)
```

SAM instruction sequence:

```
SAM Register usage:

          R0 - result low
          R1 - result high
          R2 - Vect 1 low
          R3 - Vect 1 high
          R4 - Vect 2 low
          R5 - Vect 2 high

Begin
enable(all);       {select vector region}
SAM(load,2);
SAM(andd,4);       {AND of vector low bits}
SAM(store,0);
SAM(load,3);
SAM(andd,5);       {AND of vector high bits}
SAM(store,1);
SAM(nor,0);        {look for 00}
{accumulator is set in all illegal vectors}
End.
```

Figure 5-9: Bit Vector Intersection

6. Conclusions and Future Work

In the previous sections, a description of an active bit map element has been presented. Additionally, architectures have been proposed that can control a large array of active elements. Finally, algorithms that demonstrate the parallel processing power of a synchronous active memory (SAM) have been briefly shown.

The future work in SAM consists of two primary areas: working on a complete physical implementation and the development (or modification) of algorithms. There are many options that are available for a SAM array on a chip. Possible technologies are ECL gate arrays, I^2L, CMOS and a VLSI chip in NMOS like the current prototype. For each technology, there is the corresponding tradeoff of speed, density etc. Many choices ranging from the architecture to physical chip interconnect problems need to be addressed.

The development of highly parallel algorithms is also filled many options. The final architecture and physical size of the activ map, will have an important impact. Developing algorithms that effectively partition a problem into the available bit map size connect properly into the next partition is a challenging prob Partitioning will also have a major impact on the amount of data must be moved between the physical bit map and some large bac store or alternate data representation.

The prototype chip was designed as a project by Blank and Stefik the VLSI course at Stanford taught by Alan Bell in June 1980. application of the architectures to DA problems was expa subsequently by Blank and vanCleemput.

References

1. Breuer, Melvin A. *Design Automation of Digital Systems.* Prentice-Hall Inc., 1972.

2. Duff, M. J. B. CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor. Proc. 1978 Pattern Recognition and Image Processing Conf., Pattern Recognition and Image Processing, Nov., 1978, pp. 728-733.

3. Duff, M. J. B. A User's Look at Parallel Processing. Proc. 1978 Pattern Recognition and Image Processing Conf., Pattern Recognition and Image Processing, Nov., 1978, pp. 1072-1075.

4. Lee, C. Y. "An Algorithm for Path Connections and Its Applications." *IRE Trans. Electron. Computers EC-10* (Sept. 1961), 346-365.

5. Mead, Carver A. and Conway, Lynn A. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

6. Reeves, Anthony P. and Rindfuss, R. The Base 8 Binary Array Processor. Proc. 1979 Pattern Recognition and Image Processing Conf., Pattern Recognition and Image Processing, Aug., 1979, pp. 250-255.

7. Reeves, Anthony P. "A Systematically Designed Binary Array Processor." *IEEE Trans. on Computers C-29*, 4 (April 1980), 278-287.

8. Soukup, J. Global Router. 16th Design Automation Conference Proc., IEEE Computer Society and ACM, June, 1979, pp. 481-484.

9. Unger, S. H. A Computer Oriented toward Spatial Problems. Proceedings of the IRE, IRE, Oct., 1958, pp. 1744-1750.