SADIQ M. SAIT, KHALID MAA•VLSI DESIGN AND IMPLEMENTATION OF SYSTOLIC TREE QUEUES
MICROPROCESSORS AND MICROSYSTEMS 19 (3): 139-146 APR 1995

3

VLSI design and implementation of systolic tree queues

Sadiq M Sait and Muhammad A A Khalid

A number of innovative designs have been proposed for hardware implementation of data structures. However, these designs have only been presented at an abstract behavioural level. In this paper, we describe the VLSI design and implementation of a 15-node 8-bit queue based on a systolic tree architecture. A layout methodology and a VLSI CAD environment that facilitate fast and efficient layout of large binary trees are described. The objective of this paper is to illustrate the implementation of tree architectures in VLSI. We demonstrate this by implementing a systolic tree queue.

Keywords: VLSI design, systolic tree architecture, automated layout

The advent of VLSI technology has led to the development of high performance special-purpose hardware to meet specific application requirements. Using this technology now it is possible to implement algorithms and data structures on integrated circuits (ICs). Many innovative designs have been proposed for hardware implementation of data structures[1]. However, these designs have only been presented at an abstract behavioural level. Much remains to be done in investigating the methods and strategies that employ appropriate techniques, and VLSI CAD tools, to reduce the design turn around time, and facilitate efficient physical implementation of these designs. Some of the desirable features of architectures which facilitate implementation in VLSI include modularity, regularity, and local inter-connectivity. Systolic architectures exhibit the above characteristics. For certain applications, systolic arrays exhibit higher performance than a conventional Von Neumann implementation. Systolic architectures have been proposed for several applications, for example, in digital signal processing, matrix operations, and in the implementation of data structures[1,2]. The architecture of the systolic array is characterized by the architecture of the processor, the size of the array, and the inter-connectivity of the processors. Most systolic architectures are either 1-D or 2-D arrays. Some efficient systolic architectures are also based on tree architectures and are commonly known as systolic trees. In systolic trees, the processors are nodes (also known as processing elements or PEs) of the tree, and the edges correspond to the communications paths between processors. Examples of systolic trees for data structure applications are stack, queue, priority queue, dictionary machine etc.[1,3,4]. In this paper, as an example, we choose a systolic tree queue to implement in VLSI. The methodology is applicable to any systolic tree.

The operations of a queue and its application are well known. A queue operates in a first-in first-out manner. Queues find application in several areas of computing, and in computer communication networks. Queues are also used in applications such as sorting, searching, symbol-table and decision table implementations, and in simulation studies[5].

Hardware implementations of queues

Many queue architectures have been proposed by different researchers[1,6,7]. The architecture given in Reference 6 consists of two parts, FIFO storage and a control section. The length of the queue is programmable, resulting in minimum data ripple through-times, for applications not requiring the full length of memory.

Another architecture was proposed by Kulaib[7]. This approach is similar to that of Reference 6. The one-to-one transformation of the design given in Kanopolous and Hellenbeck[6] is not practical because of the complexity of the controller. The FIFO design[7], although feasible in VLSI, is cumbersome to implement. The logic is complex and therefore a large area is required.

Tree-based architectures for VLSI implementation of queues and other similar data structures were proposed in Reference 1. In this paper we use this proposed architecture to illustrate the mapping of tree architectures in VLSI layout.

Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran-31261, Saudi Arabia. E-mail: facy009@saupm00.bitnet
Paper received: 31 December 1993

Behavioural model

The architecture of a systolic processor (in our case a single node of a systolic tree queue) is generally described at the behavioural level. The entire system architecture is a binary tree of processors, each with a unit response time and a unit pipeline interval. The root node of the binary tree is the front of the queue from which data is inserted and deleted. All nodes of the tree are identical in all respects. The behavioural model of a single processor of a binary-tree based queue[1] is given in *Figure 1*. Each (node) processor has two registers, a data register (S1) and a buffer register (B1), and some flags. The data register holds the data item to be stored in the node and the buffer register is used both as a register to store the next item of the queue, and a forwarding register, from which data is passed on to the children nodes.

The concept underlying the operation of a tree-based queue can be briefly described as follows[1]. When a node becomes active (that is, an empty node receives an insert instruction), the data item will be stored in the data register of the node, and its insert and delete flags will be set to 0. When a node contains data in its S1 register, it receives an insert instruction again, then the data item is stored in the buffer register B1 and the insert flag is complemented. Depending upon the value of other flags, the next data item will be passed on to the right or left child in the next clock cycle. Successive insert instructions will make last-in data items move to the lowest levels of the tree, that is, to the leaf nodes. When a delete instruction is applied to an active node, the data item stored in it is passed on to the parent, and the data item from its left or right child is copied into the node, and the delete instruction is passed down to the left or right child depending on the value of the flags. Basically, a delete instruction will cause deletion of a data item from the root node, and movement of data from other nodes towards the root of the tree. Similarly, an insert instruction will cause the data item to move down (eventually) to the lowest level (one of the leaf nodes) of the tree[1]. The data distribution of the queue when 15 consecutive inserts are made into the empty queue is shown in *Figure 2*.

In this paper we present the translation of the behavioural model of a single node of the queue to actual hardware in VLSI. We then describe the VLSI design of a 15 node, 8-bit queue based on the systolic tree architecture described above. The paper is organized as follows. In the next section the proposed first-in first-out (FIFO) queue design is described. We also present the algorithmic details and the model of the queue node in AHPL. This AHPL model is the hardware representation or structural RT level representation of the behavioural level model of one node of the tree-based queue. The section also discusses the synthesis of a single node in VLSI. The following section introduces the problem of tree embedding. The layout of a 15-node queue is then described. In the last section, we conclude with a discussion of our results and other possible implementations of binary-tree architectures in VLSI.

Pseudo code for Insert and Delete Instructions

```
Begin
   If (LS1 = 0 and Insert present)                                   1st condition
      Then S1 ← value; CI1 ← 0; CD1 ← 0; LS1 ← 1
   Else
      If (LS1 =1 and Insert present)                                 2nd condition
            Then  B1 ← value; CI1 ← ~CI1; LB1 ← 1
      EndIf
   EndIf
   If (LB1=LS2=LS3=0 and Delete is present)                          3rd condition
      Then LS1 ← 0
   EndIf
   If (LS2=LS3=0 and LB1=1 and Delete is present)                    4th condition
      S1 ← B1; CD1 ← ~CD1;
   EndIf
   If (~(3rd condition) & ~(4th condition and Delete present))       5th condition
      If CD1 = 0
            Then  S1 ← S2 Else S1 ← S3
      EndIf
                  CD1 ← ~CD1;
   EndIf
End.
```

Figure 1 Behavioural model of a node of a systolic-tree based queue

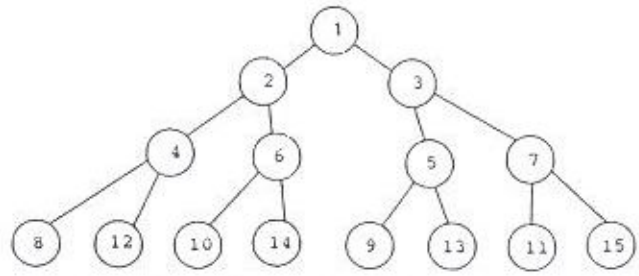

Figure 2 Data distribution of the queue after 15 consecutive inserts

DESIGN METHODOLOGY

The VLSI implementation of tree based architectures consists of the following:

1. The translation of the behavioural model of a systolic tree node into an efficient minimal structural level model.
2. The embedding of nodes of the tree into an area efficient 2-D grid.
3. The design of VLSI layout of a single node.
4. The layout of the entire tree.

Structural model

The design of the 15 node 8-bit FIFO queue is based on the architecture proposed by Chang *et al.*[1]. The first task is to translate the behavioural model of the tree node to another representation closer to hardware, i.e. an RTL model.

In this section we present the hardware design and discuss the hardware components of the node and an RT level description in AHPL[3]. A block diagram of the tree node is shown in *Figure 3*. The hardware components include two 8-bit registers, S1, B1, and eight 1-bit flip-flops LS1, LB1, CI1, CD1, IL1, DL1, IR1 and DR1. The function of each of the flip-flops is as follows. LS1 and LB1 show the status (empty or full) of data registers S1 and B1 respectively. CI1 and CD1 are insert and delete flags used for the insertion and deletion of data. IL1 and IR1 specify the direction of data movement. If IL1 (IR1) is high then the value is inserted to the left (right) child. Similarly, if DL1

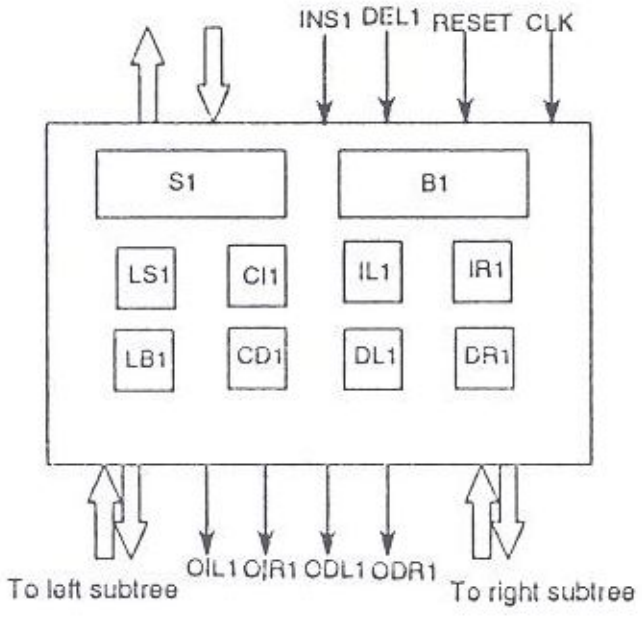

Figure 3 Block diagram of the queue node

(DR1) is high then the value from the left (right) subtree is copied to the root node and the value of the root node appears on output bus TOPD1.

There are two 8-bit lines, TOPI1 and TOPD1, for insertion and deletion of data into and from the node respectively. Four other 1-bit input buses are INS1, DEL1, RESET and CLK. If INS1 (DEL1) is high, an insert (delete) operation is performed. At the bottom of the node there are four 8-bit wide buses TOPI2, TOPD2, TOPI3 and TOPD3. The 8-bit data is inserted through TOPI2 into the left subtree, and through TOPI3 into the right subtree. The deletion of data from the left and right subtrees is routed through TOPD2 and TOPD3 respectively. There are four 1-bit output lines OIL1, OIR1, ODL1 and ODR1. If OIL1 (OIR1) is high then the data is routed to the left (right) subtree. Similarly, if ODL1 (ODR1) is high then the data from the left (right) subtree is deleted. The node has been modelled using the hardware description language AHPL[8]. The description is shown in *Figure 4*; the code corresponding to the one-state FSM is enclosed within BODY SEQUENCE and END. The node has only one state and therefore the queue instruction (insert or delete) takes only one clock cycle. The operation of a queue has been simulated using the RT level simulator available in the AHPL DA system. The simulation results have validated the design of the tree node and the queue at the register transfer level[9].

```
MODULE      : QNODE1.
MEMORY      : IL1; IR1; DL1; DR1; S1{8}; B1{8},CD1;CI1;LS1; LB1.
EXBUSES     : DEL2; INS3; DEL3;TOPI2{8},TOPI3{8}.
EXBUSES     : LSL1; LSR1; TOPD2{8}; TOPD3{8}; TOPD1{8}; OLS1; INS2; DEL2.
BUSES       : CC11; CC21; CC31; T11; T21; T31; T41; FALSE1.
EXINPUTS    : TOPI1{8}; INS1; DEL1; RESET; CLK; START.
BODY SEQUENCE :CLK.
1 CI1*(T11)    ⇐ 1$0; CD1*(T11) ⇐ 1$0;
  CI1*(T21)    ⇐ ~(CI1); LS1*(T11) ⇐1$1;
  LB1*(T21)    ⇐ 1$1; IL1*(INS1) ⇐(CI1 ! 1$0)*(LS1, ~LS1);
  IR1*(INS1)   ⇐ (~CI1 ! 1$0)* (LS1, ~LS1); LS1*(CC11) ⇐ 1$0;
  CD1*(CC21) ⇐ ~(CD1); DR1*(T31) ⇐ 1$1;
  DL1*(T41)    ⇐ 1$1; CD1*(CC31) ⇐ (~CD1); B1*(T21) ⇐TOPI1;
  S1*(FALSE1)⇐(TOPI1!B1!TOPD2!TOPD3) *(T11,CC21,T41,T31);
  TOPD1        = S1*DEL1;    ⇒ (~START)/(1).
ENDSEQUENCE
CONTROLRESET (1);
  TOPI2 = B1*IL1; TOPI3 = B1*IR1;
  FALSE1= (T11+CC21+T41+T31);
  INS2= IL1; DEL2= DL1; OLS1= LS1; INS3= IR1;
  DEL3= DR1; T11 = INS1 & ~LS1;
  T21 = INS1 & LS1; T31 = CC31 & (~CD1); T41 = CC31 & CD1;
  CC11 = ~(LSR1 + LSL1 + LB1) & DEL1;
  CC21 = ~(LSR1 + LSL1 + ~LB1) & DEL1;
  CC31 = ~(CC11) & ~(CC21) & DEL1.
END.
```

Figure 4 AHPL description of systolic tree-based queue node

Logic synthesis

Hardware description languages (Verilog, CDL, DDL, AHPL, VHDL, ISPS etc.) have been used successfully for documentation, communication and verification[10]. They have also been used as input specification languages to design automation (DA) systems which synthesize VLSI layouts. The logic synthesis of the node of a systolic queue is done with the help of a hardware compiler[8]. After the AHPL code is compiled, it is simulated at RTL level with the help of a functional simulator, and the logic synthesis is

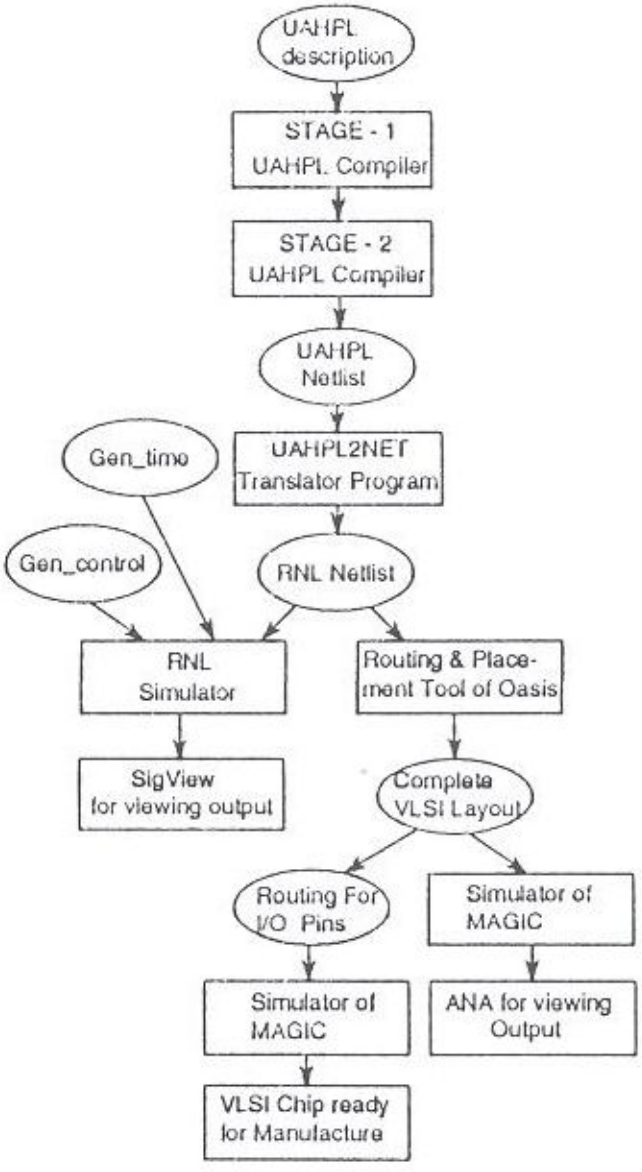

Figure 5 An overview of the DA system for generation and verification of hardware

carried out. It generates a logic netlist of the hardware circuit. The netlist comprises combinational logic gates and D flip-flops.

The AHPL netlist is translated to the RNL compatible logic level netlist with the help of a translator written in the C programming language. The netlist generated is simulated by RNL[11], a timing and logic level simulator. The simulation results of RNL are observed using SigView, a simulation (signal) viewing tool. The process is illustrated in *Figure 5*.

Layout of a node

After generation of the netlist for the single node in RNL format, the layout sub-system of OASIS called Vanilla Place aNd Route (VPNR)[12] is used to translate the netlist into a VLSI layout. The process of translating netlists to layouts is also illustrated in *Figure 5*. The standard cell library of OASIS consists of scalable CMOS cells compatible with 2 μm SCMOS technology of MOSIS. VPNR, in addition to generating the standard cell layout from the RNL netlist, incorporates testability by including scan path based testing circuitry in the layout. It also performs consistency checking of routing and placement phases of the design. The complete VLSI layout of one node is shown in *Figure 6*. The area of the layout is 1.53 mm^2.

As a first step in layout verification, the circuit representation of the VLSI layout is extracted using Magic's circuit extractor and simulated using *irsim*, a switch/transistor level simulator[13]. The results of simulation are observed on a viewing tool 'ana' and are illustrated in *Figure 7*.

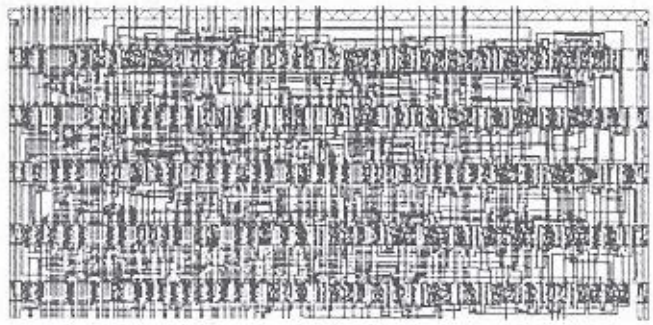

Figure 6 Layout of one node

TREE EMBEDDING

In VLSI system design, it is most important that the final layout obtained has the desired aspect ratio. It is also important that dead space on the VLSI layout be at a minimum. Therefore, a scheme is required to map the nodes of a binary tree onto a rectangular grid so that the amount of dead space is minimized. In addition, to avoid degradation in performance, it is preferable that strongly connected nodes (parents and children of the tree) are placed as close to each other as possible.

Several schemes are available in the literature to map a binary tree into a 2-D grid of processors. This mapping is referred to in the literature as 'tree embedding'. Important schemes available for tree embedding are given below.

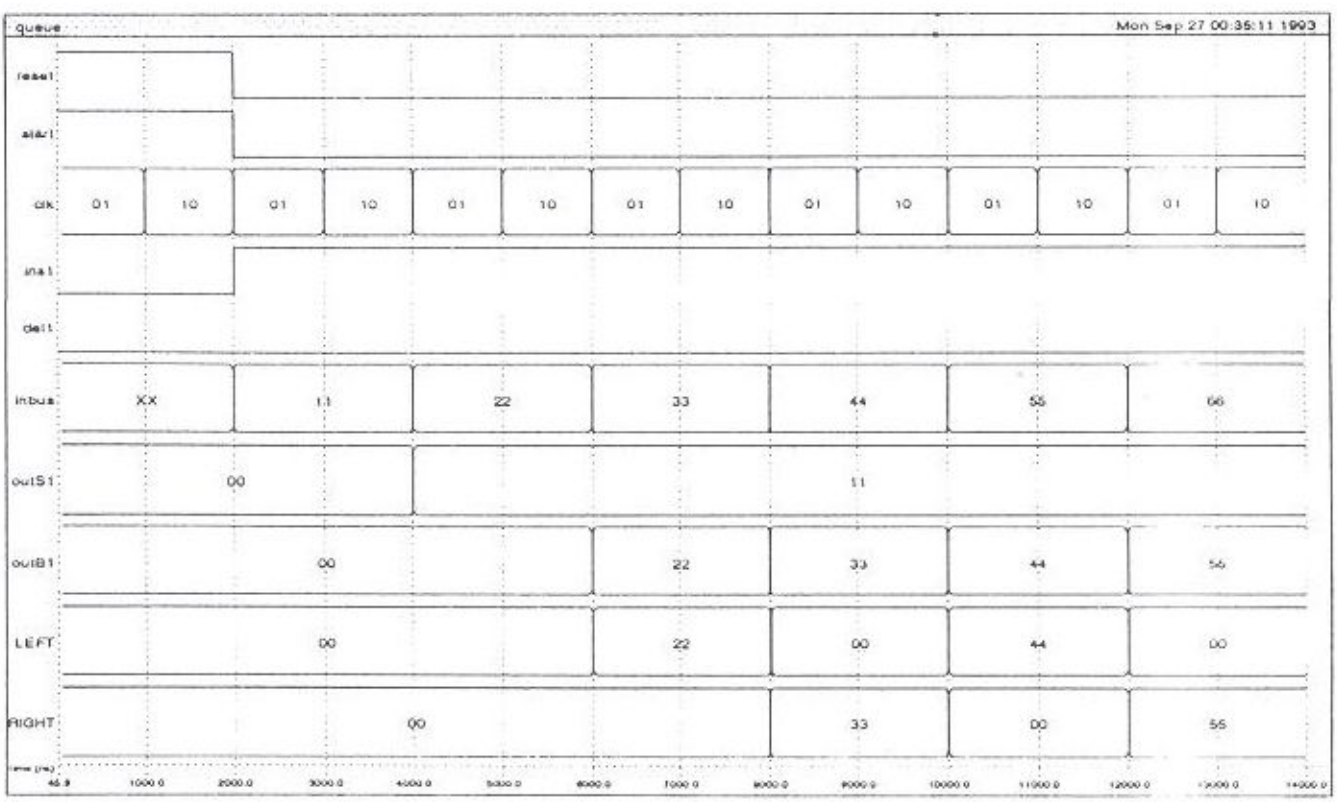

Figure 7 Transistor-level simulation of one node

H-trees

H-trees represent a convenient way to place the nodes of a complete binary tree on a grid (it has appeared in many references, the earliest being apparently References 3 and 14). The placement can be done recursively, and the final pattern looks like the letter 'H', hence the name. The nodes of the binary tree can be represented in a layout by grid points. An H-tree of order i represents a binary tree of height $2 \times i$ (see *Figure 8*). Horowitz and Zorat[15] presented a detailed algorithm for mapping the binary tree in the form of an H-tree. As seen from the figure, this form of embedding is not area efficient. Also the length of interconnects is large, resulting in significant propagation delays. This scheme is only 50% area efficient, much of the chip area being wasted. The advantage, however, is in ease of placement (recursively) and ease of routing. Bounds on edge lengths and areas for embedding of trees with leaves on the border of the grid, and a plan of the binary tree layout with low maximum wire length, are presented in Reference 2. Clearly, the above scheme is not suitable for our embedding because of low area efficiency and high interconnection lengths[2, 15].

Hexagonal tree embedding

Another approach to solving the problem of embedding a complete binary tree in a square or hexagonally connected VLSI array of processing elements was studied by Gordon[16, 17]. The scheme is different from other techniques. Here, instead of considering the problem as that of laying out a graph in a plane, PEs are used both as tree nodes and as connecting elements between distant nodes. This approach shows a slight improvement over a previous approach that uses H-trees[15] in terms of utilization of chip area and propagation delay. It is 71% area efficient[16, 17]. It can also be seen from *Figure 9* that the nodes shown as squares and all surrounding nodes are wasted, creating longer edges and hence a larger propagation delay[17].

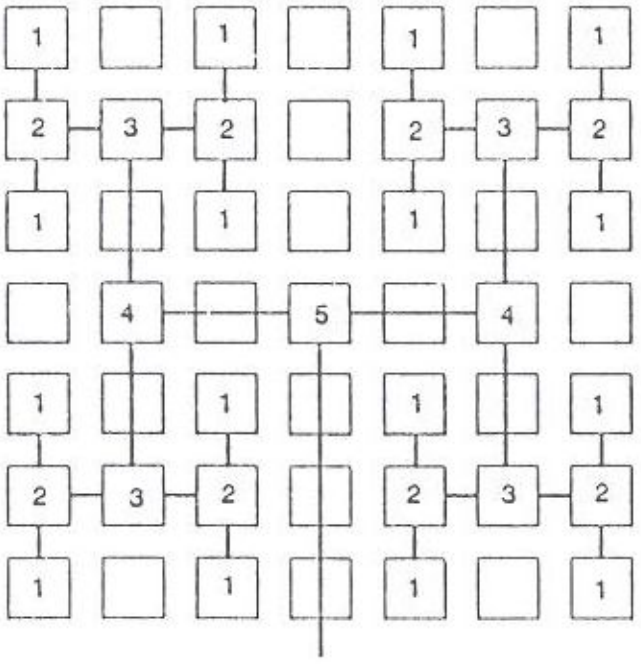

Figure 8 A five-level tree embedding in H-tree layout

Tile-based tree embedding

An efficient scheme for the layout of large binary tree architectures by embedding the complete binary tree in a 2-D array of PEs was presented by Youn and Singh[18]. This scheme, also known as 'tile based design', utilizes virtually 100% of all PEs in the array, and also shows substantial improvement in propagation delay and maximum edge length over H-tree layouts. In addition, the layouts produced readily lend themselves to fault-tolerant designs for overcoming fabrication defects in large areas. As an example, the 4-level (15 node) binary tree embedding is shown in *Figure 10*.

Implementation of larger queues

The scheme suggested by Youn and Singh[18] for embedding larger trees uses layouts of four different modules, M1 to M4, which are created and stored. Each module M_i is an embedding of a four-level binary tree similar to that in *Figure 10*. The four modules are different from one another in the relative position of the unused node with respect to the root node of the four-level subtree. The four types of basic modules are shown in *Figure 11*. The number in the box (node) denotes the level of the node in the four-level binary tree. Note that the modules have to be rotated/reflected depending on where they are used. Whenever larger trees are constructed using these basic modules, the root node of the tree should be located at the centre of the two-dimensional host array for minimum propagation delay. More details of efficient embedding of large binary trees are given in Reference 18.

Larger queues of any length can be laid out and efficiently connected together without incurring a significant area penalty. The connection strategy for different level trees is as follows:

- Five-level trees: A five-level tree can be embedded by laying out and connecting one M1 basic module and

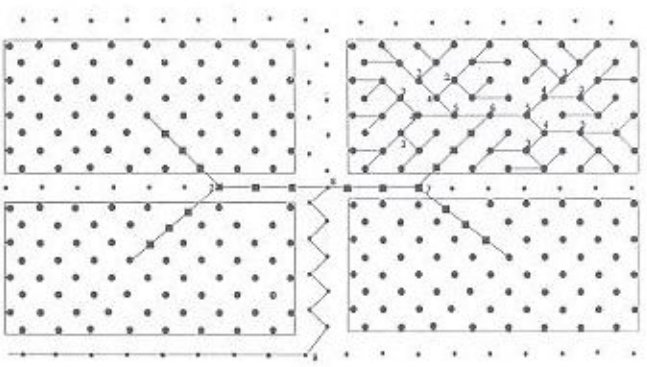

Figure 9 The regular embedding scheme for hexagonal arrays[16]

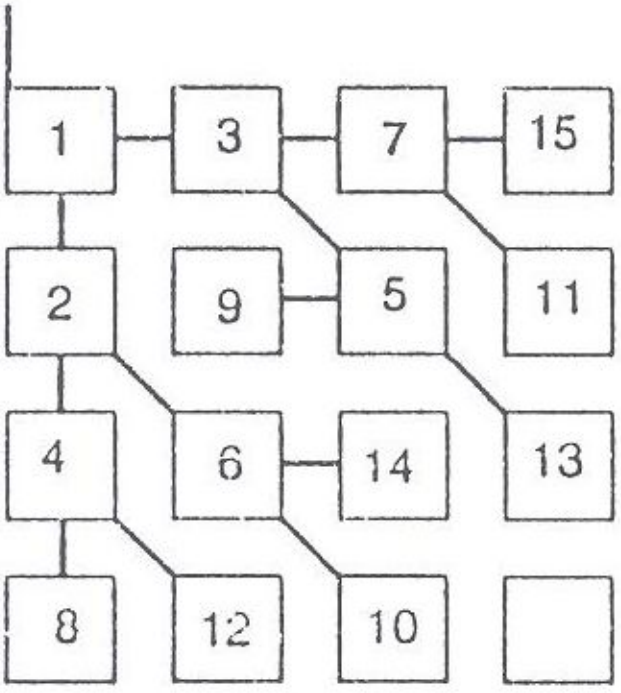

Figure 10 Embedding of 15 node tree in a 4 × 4 square array of processors

one M3 basic module side by side. The unused node in the basic module M1 is utilized as the root node of the five-level tree. Actually, one M1 basic module with either an M1 or M2 basic module can also be used to construct a five-level tree. However, these combinations are not preferred because they lead to longer propagation delay.

- Six-level trees: A six-level tree can also be constructed with two M1, one M3 and one M4 basic module. Other combinations of four basic modules such as four M1, or three M1 and one M3 can also produce the six-level tree. Again these combinations are not preferred due to the longer propagation delay. The basic module M4 is used only once in the six-level tree embedding for achieving minimum propagation delay.
- Trees of greater than seven levels are obtained by repeatedly taking two trees of a given size and constructing a larger tree with one additional level. This process can be repeated beginning with two six-level trees until a tree of the desired size is obtained.
- To get an $n+1$ level tree from an n level tree, we proceed as follows:
  - Obtain the mirror image of the n level tree along the edge containing the unused processor.
  - Combine the two layouts and locate the root of the $(n+1)$th level tree in the unused processor in the original n level tree.
  - Rotate the set of modules (in their correct relative position) about an axis parallel to the appropriate array edge so as to move the unused processor to the outside of the array.
- This process can be repeated recursively until the desired size is obtained.

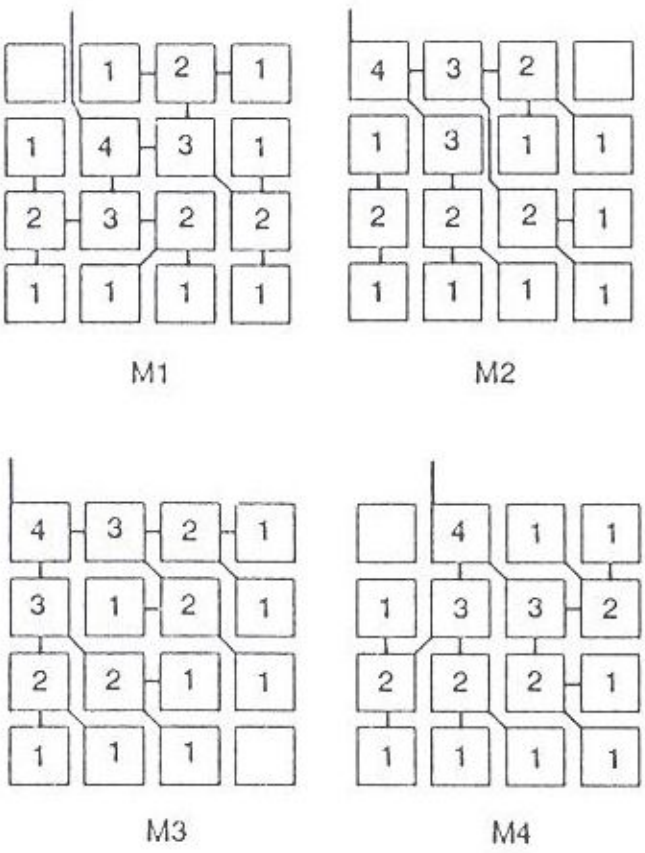

Figure 11 Structure of basic modules used for embedding trees with level greater than four[18]

LAYOUT AND SIMULATION

From the discussion above it is clear that currently, the tile-based layout design is the best approach for embedding binary trees into a 2-D grid. The completely routed VLSI layout of 15 nodes that uses the above embedding is shown in *Figure 12*. Fifteen out of 16 squares in the grid are utilized as FIFO queue nodes. The remaining unused node in each basic module is used as a tree node at some higher level, when the basic modules are connected to build a larger tree.

The generated VLSI layout of the single node (see *Figure*

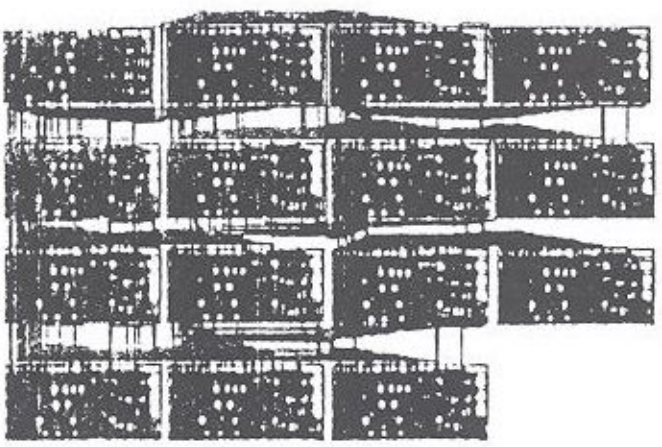

Figure 12 Layout of 15-node queue

6) has dual I/O terminals, both at the top and bottom of the node layout. The router used to interconnect modules decides which of the pins, i.e. those at the top or from the bottom, are to be chosen for interconnection.

Since the architecture is systolic, to verify the correctness of the design, in our case, it is sufficient to demonstrate the simulation of three nodes of the queue. The layout of the first three nodes, that is, a root and two children, is extracted, and the extracted circuit is simulated to verify the functional correctness. Simulation output of a three-node queue is given in *Figure 13*. The entire queue is also simulated for functional correctness. Observe that when insert is high, five data items enter the queue, that is, 11, 22, 33, 44 and 55. Since there are only three nodes, the first three items stay in the S registers of the three nodes, respectively. When the delete signal goes high, the data items leave the queue in the FIFO order, item 11 appears on 'outbus', then 22 and then 33. The area of a 15×8 systolic queue is 33.9 mm^2. The queue operates at a clock rate of 18 MHz.

DISCUSSION AND CONCLUSION

The design and VLSI implementation of a 15×8-bit FIFO queue, based on a systolic tree architecture, has been presented. The design is modular and can be extended to a queue of larger size. This work provides insight into the issues involved in physical VLSI implementation of architectures based on binary trees. Our experience with this layout confirms that such architectures have very efficient VLSI implementations, and are suitable for automated layout. The layout methodology relies on the following VLSI design tools: (a) the AHPL DA system for RT level modelling, functional simulation and netlist generation, (b) RNL for logic level simulation, (c) the VPNR subsystem of OASIS for placement and routing of standard cells, and (d) MAGIC for layout, routing, circuit extraction from layout, and simulation of extracted circuit (using MAGIC's *irsim*). Design rules for layout and the technology files for simulation used were obtained from MOSIS. A MOSIS pad frame is used to complete the layout for fabrication.

The logic/transistor level simulation results obtained using RNL/*irsim* were observed using viewing tools SigView/ana, respectively. The simulation results were in conformity with the functional level simulation results. This verification greatly enhances the chance that the chip will work correctly.

The total area of the 15-node layout is 33.9 mm^2, and of this 30% is used by the router (including dead space). Larger width queues can be constructed easily by modifying the sizes of buses and registers in the RT-level model. The bus-width is limited by the I/O pins on the IC package. The entire process can be easily automated. For design of other data structures such as stack, only the RT level model has to be modified. The overhead in terms of number of flags and buffer registers per node is high, but the ease of implementation (modularity, local interconnectivity, etc.) justifies designing ICs using such architectures.

The work presented can be continued on practical VLSI realization of such architectures proposed for a variety of applications. Examples include systolic tree architectures for other data structures such as stack, dequeue, priority

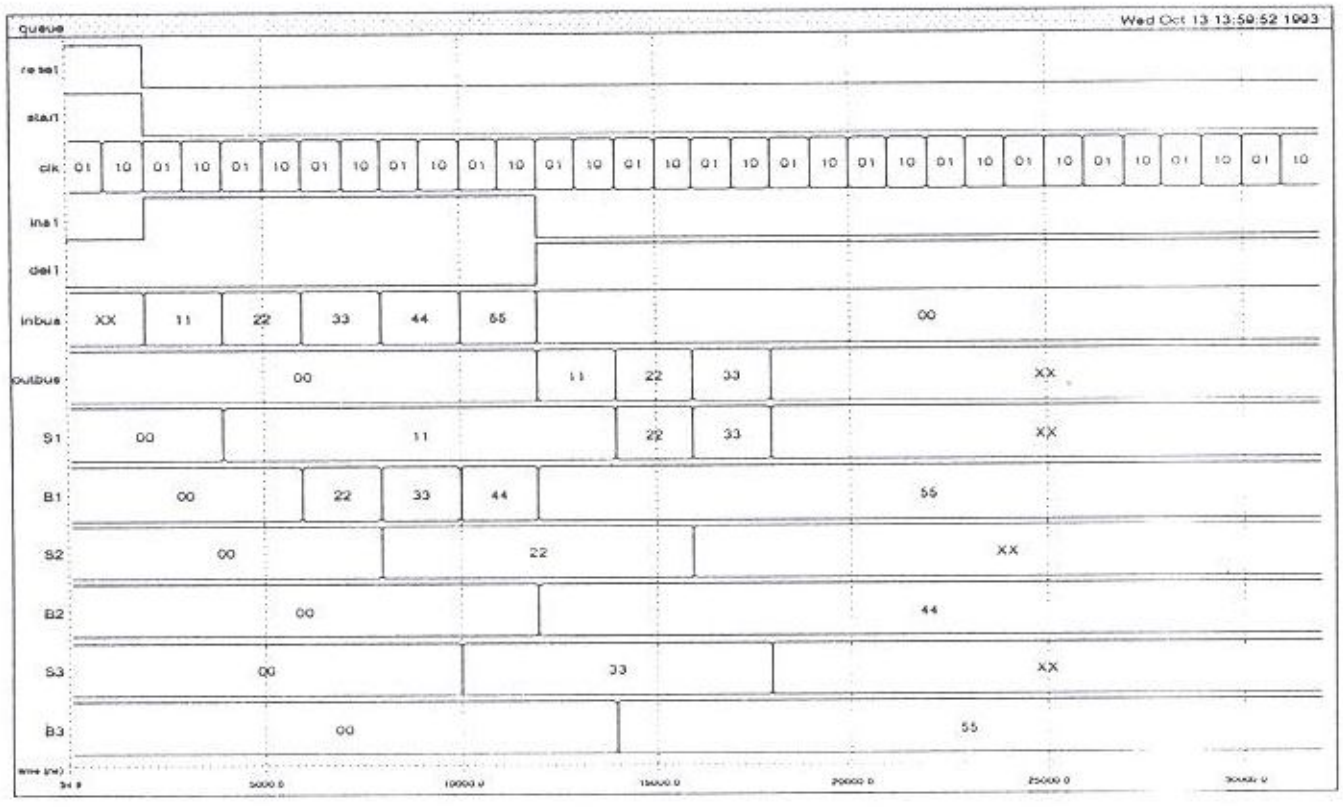

Figure 13 Transistor-level simulation of three nodes

queue, dictionary machine[1], systolic tree implementation of data compression algorithms[19], etc. Our experimental studies show that the tile-based approach is most area efficient[18]. Layouts of 15-node queues are easily expandable to queues of larger sizes.

ACKNOWLEDGEMENTS

The authors acknowledge King Fahd University of Petroleum and Minerals for support of this work. We would like to thank Dr H Youssef and Dr M Abd-El-Barr for their comments and discussion that helped to improve the quality of this manuscript.

REFERENCES

1 Chang, J H, Ibarra, O H, Chang, M J and Rao, K K 'Systolic tree implementation of data structures' *IEEE Trans. Comput.* Vol 37 No 6 (1988) pp 727–735
2 Ullman, D J *Computational Aspects of VLSI* Computer Science Press (1987)
3 Leiserson, C 'Systolic priority queues' Tech. Rep. CMU-CS-79-115, Dept. Comput. Sc., Carnegie Mellon University (1979)
4 Browning, S A 'Computations on tree of processors' *Caltech Conference on VLSI* (January 1979) pp 453–478
5 Kruse, R L *The Art of Computer Programming* Prentice-Hall India (1987)
6 Kanopolous, N and Hellenbeck, J J 'A first-in, first-out memory for signal processing applications' *IEEE Trans. Circ. Syst.* Vol CAS-33 (May 1986) pp 556–558
7 Kulaib, M A *et al.* 'Design of a programmable length FIFO memory and its controller' *Int. J. Electronics* Vol 65 (1988) pp 923–932
8 Masud, M and Sait, S M 'Universal AHPL – A language for VLSI design automation' *IEEE Circ. Dev. Mag.* (September 1986)
9 Sait, S M 'Integrated UAHPL-DA system with VLSI design tool to support VLSI DA courses' *IEEE Trans. Educ.* (September 1992)
10 Hill, F J 'AHPL Then and now' *IEEE Des. Test Comput.* (June 1992) pp 73–75
11 *VLSI Design Tools Manual* NW Laboratories for Integrated Systems, Release 3.1 (February 1987)
12 MCNC Open System Silicon Implementation Software, reference manual (1992)
13 Ousterhout, J K *et al.* 'The MAGIC VLSI layout system' *IEEE Des. Test* (February 1985)
14 Bentley, J and Kung, H T 'A tree machine for searching problems' *Proc. Int. Conf. Parallel Processing* (August 1979) pp 257–266
15 Horowitz, E and Zorat, A 'The binary tree as an interconnection network: Application to multiprocessor systems and VLSI' *IEEE Trans. Comput.* Vol C-30 (April 1981) pp 247–253
16 Gordon, D *et al.* 'Embedding tree structures in VLSI hexagonal arrays' *IEEE Trans. Comput.* Vol C-33 (1984) pp 104–107
17 Gordon, D 'Efficient embedding of binary trees in VLSI' *IEEE Trans. Comput.* Vol C-36 (1987) pp 1009–1018
18 Youn, H Y and Singh, A D 'On implementing large binary tree architectures in VLSI and WSI' *IEEE Trans. Comput.* (1989) pp 526–537
19 Thompson, C D and Wei, B W *Systolic Implementations of a Move-to-Front Text Compressor* ACM (1989)

Sadiq M Sait obtained a Bachelor's degree in electronics from Bangalore University in 1981, and Master's and PhD degrees in electrical engineering from King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, in 1983 and 1987 respectively. Since 1987 he has been working at the Department of Computer Engineering, teaching graduate and undergraduate courses in the areas related to digital design automation, VLSI system design and computer architecture. His current areas of interest are in digital design automation, VLSI system design, and high level synthesis.

Mohammed Abdul Aziz Khalid was born in Hyderabad, India, on December 23, 1968. He received a B.E. degree in computer science and engineering from Osmania University, Hyderabad, India in 1990. He joined King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, as a Research Assistant in Fall 1991, and obtained an MS degree in computer engineering from there in 1994. Currently he is working as a systems analyst at the Data Processing Center of the university. His areas of interest include VLSI design automation, computer architecture and computer communication networks.