The Transmogrifier-2: A 1 Million Gate Rapid-Prototyping System

David M. Lewis, *Member, IEEE,* David R. Galloway, *Member, IEEE,* Marcus van Ierssel, Jonathan Rose, *Member, IEEE,* and Paul Chow, *Member, IEEE*

***Abstract*—This paper describes the Transmogrifier-2 (TM-2), a second-generation multifield programmable gate array (FPGA) rapid-prototyping system. The largest version of the system will comprise 16 boards that each contain two Altera 10K50 FPGA's, four I-Cube interconnect chips, and up to 8 Mbytes of memory. The inter-FPGA routing architecture of the TM-2 uses a novel interconnect structure, a nonuniform partial crossbar, that provides a constant delay between any two FPGA's in the system. The TM-2 architecture is modular and scalable, meaning that systems of various sizes can be constructed from copies of the same board, while maintaining routability and the constant delay feature. Other features include a system-level programmable clock that allows single-cycle access to off-chip memory, and programmable clock waveforms with edge resolution of 10 ns. The first Transmogrifier-2 boards have been manufactured and are functional. They have recently been used successfully in some simple graphics acceleration applications.**

***Index Terms*— Architecture, field programmable gate array (FPGA), memory, Rapid prototype, scalability.**

I. Introduction

CONTINUING advances in the density and speed of field programmable gate arrays (FPGA's) have made them effective implementation vehicles for increasingly complex systems. Nevertheless, contemporary FPGA's lag semicustom application specific integrated circuits (ASIC's) by a factor of ten or more in density, and lack system-level facilities, such as large random access memories (RAM's) and clocks, that are needed to implement large systems. Multi-FPGA field-programmable systems can be used not only to prototype these larger designs, but also as field-configurable compute accelerators. A number of field-programmable systems have been described [1]–[7], and can be roughly classified either as emulation systems, or custom compute engines. Emulation systems tend to be targeted at emulating ASIC's, which can be designed for a wide range of applications. These systems can implement a large variety of circuits, but suffer from low clock rates and poor logic utilization. Custom compute engines tend to have architectures that constrain the applications to those that fit the computational model offered by the hardware The application must be designed specifically for implementation on the target architecture, but can achieve high clock rates and good FPGA utilization as a consequence, provided that the design can be mapped onto the architecture.

The Transmogrifier-2 (or TM-2) is a flexible rapid-prototyping system that offers high capacity and high clock rates, and is flexible enough to implement a wide variety of systems. This system offers a higher degree of flexibility than previous compute engines, although it is expected that a design to be implemented on the TM-2 is designed with the TM-2 in mind as a target, as opposed to being a verification engine for designs that will later be implemented in semicustom or full-custom silicon. By designing with the TM-2 as a target, it is hoped that higher utilization than emulation engines can be attained, while with the inclusion of a flexible routing structure it is hoped to be possible to accommodate a wider range of applications than compute engines.

We have previously constructed a small-scale rapid-prototyping system, the Transmogrifier-1 (TM-1) [7]. It contained 40 K FPGA gates and 128 Kbytes RAM, and was capable of implementing small systems. Constructed from a commercial rapid-prototyping board plus other components and software, the TM-1 was difficult to use and required that the user be physically present to operate the machine. A number of designs were successfully implemented on the TM-1, with our positive and negative experiences largely leading to the goals for the TM-2 project.

The remainder of this paper describes the goals of the TM-2 project and the resulting architecture and design. Section II describes the goals of this project. Section III describes the routing architecture developed for the TM-2. In Section IV, the influence of the goals and technical constraints on the detailed design are described. The software development system for the TM-2 is briefly described in Section V, and Section VI provides a summary of the project's current status.

II. Goals

Our experience with implementing applications on the TM-1, and the desire to create an effective, large-scale and easily usable system led to the following goals for the TM-2.

1) *Modularity and Scalability:* The terms modular and scalable refer to two related, but distinct properties. By scalable, we mean that the architecture should be usable in a range of sizes, up to some upper bound. The term modular refers to the way in which these systems are constructed, and means that all systems are built from a number of identical modules. Each module is a

Manuscript received March 28, 1997. This work was supported by NORTEL Technologies, the Natural Sciences and Engineering Research Council of Canada, and Ricoh Corporation.

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Ont., M5S 3G4 Canada.

Publisher Item Identifier S 1063-8210(98)02954-0.

single FPGA together with sufficient routing resources to provide all interconnect to the rest of the system, as well as other associated hardware, such as memories.

2) *1 Million Gate Capacity, Modern FPGA's:* The system should be configurable up to 1 M gates plus RAM, and use modern FPGA's to achieve this capacity. The TM-2 uses the Altera 10K50 [11], with 35 K usable logic gates (not including embedded RAM). A 32-FPGA system exceeds 1M gates.
3) *Well-Behaved Interchip Routing Delay:* Routing delay can be the primary performance limit in a field-programmable system. In a scalable system such as the TM-2, we would like the routing delay to grow as slowly as possible as the size of the system increases. Furthermore, the routing delay should be easily predictable for any size of system.
4) *High Speed:* The TM-2 should be capable of implementing systems with "reasonably" high user clock rates, on the order of 10 MHz when design is performed with little attention to the properties of the TM-2. We expect that higher clock rates will be achievable when the user optimizes the implementation for the specific architecture of the TM-2.
5) *User Programmable, High-Quality Clocking:* The system should be able to generate an adequate number of clock signals of arbitrary frequency and duty cycle, and to establish phase relationships between multiple clocks. The clocks should be fully programmable on a cycle by cycle basis. This feature is useful for generation of high-quality write enables for commodity asynchronous RAM's, which must be precisely controlled, but not asserted every cycle. Clock frequency and edge resolution should be fine enough that there is no significant impact on system performance. System-wide skew must be negligible. It should also be possible to derive other frequency or phase-related clocks from external signals.
6) *High RAM and I/O Bandwidth:* The system should have large amounts of high-bandwidth RAM, with high-speed access. Because RAM is relatively inexpensive compared to FPGA's and total system cost, we prefer to provide an excess amount of RAM, with the consequence that most of it will not be used in typical applications, if this has any advantage in increasing speed or decreasing routing chip requirements. Similarly, the system should provide abundant I/O to the external logic. I/O must scale with logic capacity. The TM-2 includes abundant SRAM or DRAM, with a large I/O capacity.
7) *User Friendliness:* The system should support a fully automated CAD flow. It should be possible to compile from a high-level design down to configuration bit streams for the individual components with a single command. Hardware and software support should be provided for debugging, so an adequate number of signals can be observed without recompiling the design. It should be possible to access the TM-2 across a network, including downloading, usage, status monitoring, and debugging, without needing to be physically present to perform reset operations. The TM-2 provides hardware facilities for debugging, as well as core software to support the creation of software interfaces to it.
8) *Programming Time:* Download into the TM-2 should be fast, under 10 s.
9) *Reliability:* The TM-2 should be constructed in an electrically and physically robust manner. The system should prevent defective designs from causing destructive overcurrents in chips, with possible chip failure as a result. The TM-2 includes interlocks to prevent hardware failures.
10) *Manufacturability:* The TM-2 must be straightforward to manufacture; low-cost, standard PCB technology and commercial, off-the-shelf IC's should be used exclusively. No exotic signalling should be used for the interconnect, although this is a tempting digression given the massive number of signals in such a system.

III. Routing Architecture

The reconfigurable inter-FPGA interconnect architecture is the key aspect of the TM-2 that determines system performance. We begin by discussing the effect of the goals in Section II on the architecture.

A. Requirements and Overview

The requirement for a modular and scalable system, together with manufacturability constraints, leads us to an implementation that consists of a number of identical PCB's plugged into a backplane that provides interconnect between the PCB's. Each PCB contains one or more modules, where a module is defined as a single FPGA and some reconfigurable interconnect. The backplane is assumed to be passive, containing only a fixed set of connections between the slots. This implementation leads to the primary design difficulty: the potentially large number of backplane wires required to provide interconnect between all modules. The key limitation in fabrication is the maximum number of wires at any point in the backplane, and it is this number that our routing architecture attempts to reduce. The goal of good system speed also restricts the choice of interconnect structures to those with delay that grows slowly with increasing system size, and has an easily predicted upper bound.

The following assumptions concerning the qualities of the FPGA's, manufacturability concerns and usage of the system by users also guide the architecture.

- The pin assignment on the FPGA's can be imposed by the software system—i.e., both routability and speed of the individual FPGA's is maintained in the presence of arbitrary pin permutations. Studies presented in [10] suggest that fixed pin assignments on contemporary FPGA's have small influences on routability and speed.
- System I/O signals are assigned to specific pins as specified by the user, and cannot be altered by the TM-2 compilation. This allows the user to plug the TM-2 into some other system without rewiring every time the design is recompiled.

- A single module contains both the FPGA on a PCB, and any programmable interconnect to support the FPGA's on the PCB in any sized system. (Here size refers to the number of modules).

A manufacturing goal for the backplane is to have moderate pin counts on the modules, and wiring density as low as possible.

Our initial analysis, and the discussion presented in this paper assumes that the FPGA has 300 pins, although the Altera 10K50 actually has 304 usable signal pins. We use 300 pins in this paper as it is a round number.

The simplest possible structure is a full crossbar. This is clearly infeasible as a system containing 32 FPGA's, each of which has 300 signal pins would require $32 \times 300 = 9600$ wires from the FPGA's to the crossbar, clearly an excessive level of complexity. A full crossbar is not scalable or modular, as the amount of routing hardware grows as $O(n^2)$, so the size of each module grows as $O(n)$.

The next simplest routing architecture is a partial crossbar, also related to a folded Clos network [1], [8]. A partial crossbar network converts a large crossbar into a collection of smaller crossbars, arranged as three stages of crossbars. In field-programmable systems, the two outer stages can be merged into the FPGA's, provided that the FPGA's can be routed with any assignment of I/O signals to the pins of the FPGA. The network is folded about the middle, so the inner stage can be implemented in a single physical crossbar. This architecture requires a collection of smaller crossbars, the size of which is an integral multiple of the number of FPGA's in the system. The complexity of this system grows as $O(n^2)$, but since each of the crossbars needs only a moderate number of pins, the total routing hardware complexity is acceptable. It is possible to include sufficient crossbar capacity on each module to accommodate the largest desired system, but the backplane wiring count for systems of 32 FPGA's is still 9600 and would be difficult to manufacture. However, the partial crossbar is a valuable idea that is used in the design of the hierarchical routing architecture for the TM-2. Separate crossbars are required for the I/O signals if it is desired to be able to assign each I/O signal to a specific I/O connector pin.

The TM-2 routing structure uses the partial crossbar structure, but modifies it to take advantage of the natural hierarchy and resulting locality of wiring within circuits. Fig. 1 shows a partial crossbar architecture, containing four FPGA's and two I/O modules. The internal routing of the FPGA provides one stage of crossbar, and separate routing chips interconnect the FPGA's. This figure illustrates the use of locality in routing by adjusting the number of signals between each FPGA, with heavier lines representing a larger number of wires. In Fig. 1, the circuit has a higher degree of local connectivity, represented by the thicker lines, between FPGA's A and B, and between C and D. Each FPGA is assumed to be physically close to a crossbar with the same label. The varying degree of connectivity between FPGA's is reflected in the different number of wires between these FPGA's and physically nearby crossbars. The local wiring could be distributed across all of the crossbars, but concentrating it in physically nearby crossbars reduces the maximum wiring density of the backplane, although the total number of wires is constant.

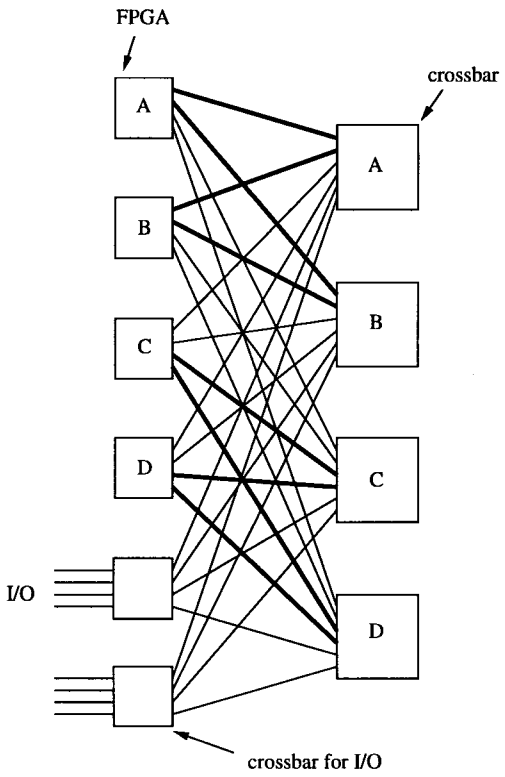

Fig. 1. Partial crossbar with nonuniform routing density. Thicker lines have a larger number of wires.

B. Relation of Wire Counts to Hierarchical Bipartitioning

This section shows how the wire counts from each FPGA to each crossbar can be related to the cut counts in a recursive bipartitioning of a network. This principle is illustrated by means of an example, with typical wire counts chosen. Assume that we are given some circuit, and wish to implement it in four FPGA's. To do so, we will recursively bipartition the network to fit in the FPGA's. Fig. 2 illustrates how the wiring can be distributed across the crossbars in a manner that directly results from the hierarchical partitioning of a circuit. Fig. 2(a) shows the result of the first bipartitioning of the circuit, resulting in 120 signals that cross the top-level cut. In Fig. 2(b) the second bipartitioning of each of the left and right partitions splits these 120 signals into 60 for each of the new partitions, and induces another 100 nets between each of the new partitions. Fig. 3(c) demonstrates how three crossbars could be used to implement this interconnect. The total connectivity of three crossbars can be split into eight smaller partial crossbars by dividing the 240 pin crossbar into four, and each of the 200 pin crossbars into two pieces, as shown in Fig. 2(d). Next, in Fig. 2(e) these are grouped together so that each FPGA has an associated crossbar. It is not necessary to group all of the partial crossbars related to a single module into a single physical crossbar chip, although this does have some desirable properties that will be mentioned later. Finally, Fig. 2(f) shows a modular structure that contains a single FPGA and a 160 pin crossbar $(65 + 65 + 15 + 15)$

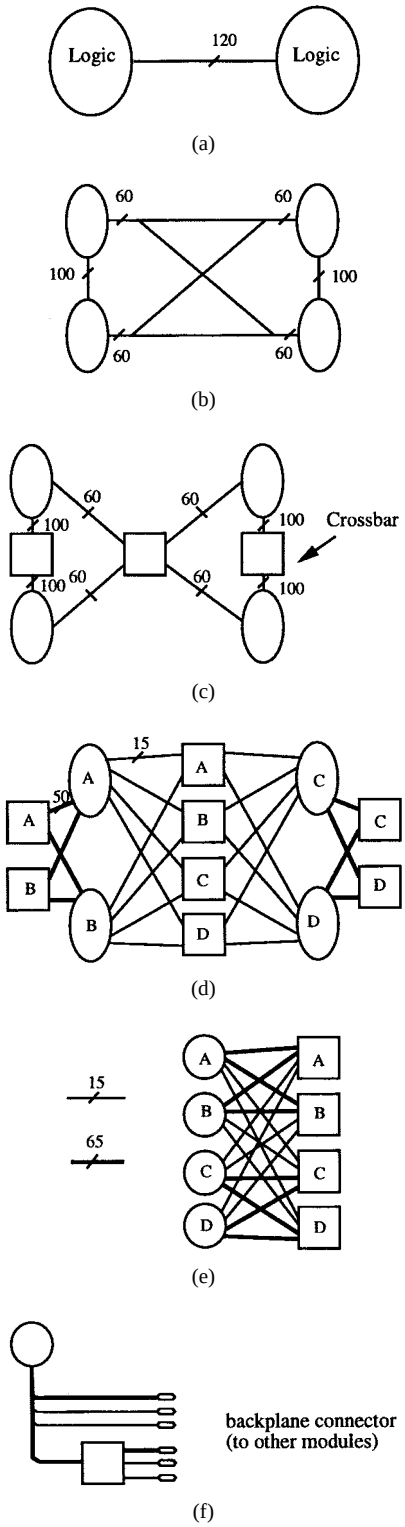

Fig. 2. Construction of nonuniform partial crossbar using bipartitioning.

implementing the two separate partial crossbars. Backplane wiring according to the density in Fig. 2(e) is used to connect the modules. While this example shows all partial crossbars merged into a single physical crossbar device, there could be multiple physical crossbars corresponding to routing at the various levels of hierarchy or even multiple physical crossbars for a single level of hierarchy.

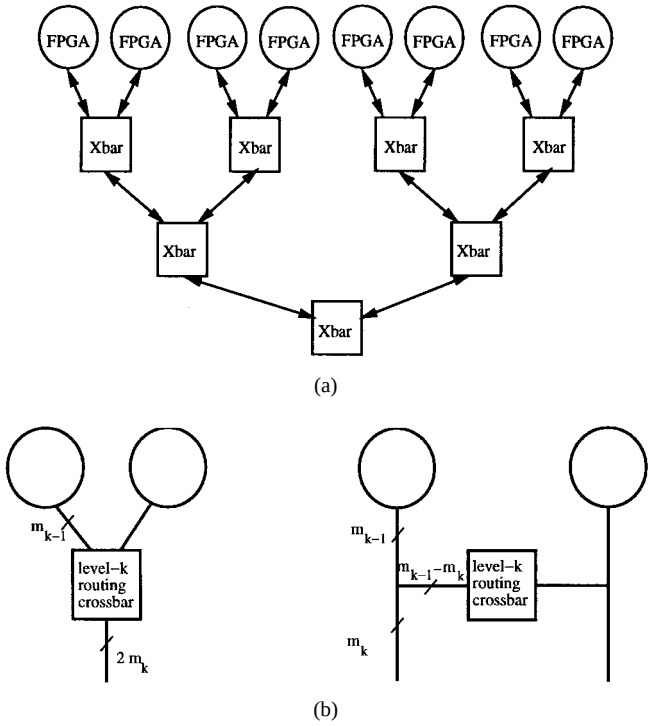

Fig. 3. Flattening a hierarchical routing architecture. (a) Hierarchical routing architecture. (b) Locally flattening the routing.

This network can be implemented with reduced backplane wiring density, compared to a uniform partial crossbar. As shown in Fig. 2(e) the maximum wiring density in the system occurs between modules A and B, or C and D, and is 190 $(65 \times 2 + 15 \times 4)$ wires. A uniform partial crossbar would have the maximum wire density between modules B and C, and require a maximum density of 320 (40 $\times$ 8).

C. Derivation of Crossbar Sizes

The above example has assumed specific values for the number of interconnect wires required at each level. The next step is to determine the wiring requirements for a nonuniform partial crossbar according to the typical routing requirements of a recursively bipartitioned circuit. Rent's rule, which predicts the total amount of wiring that leaves a system of a given size, can be used to predict the number of cuts induced when a circuit is bipartitioned.

A circuit implemented in the TM-2 can be viewed as a hierarchy of networks, with a level-k system comprising 2^k FPGA's. A level $k+1$ system is constructed with 2 level-k systems and some programmable interconnect. Conceptually, a level-k system is constructed by using a level-$k+1$ crossbar to interconnect two level-k systems. In reality, the technique demonstrated in Fig. 2 is used to implement all levels of the hierarchy at a single level of routing. Fig. 3 illustrates the relationship between a hierarchical routing architecture and our routing architecture. Fig. 3(a) illustrates a hierarchical routing architecture. Fig. 3(b) illustrates the number of pins at each level, and how this can be locally transformed into a flat routing architecture. If there are m_k pins per FPGA at level k, then m_{k+1} of these pass through level k and up to level $k+1$.

The crossbar serves no purpose for these pins, so it is possible to transform it into the flat structure, that directly connects m_{k+1} to the next level, and connects $m_k - m_{k+1}$ at level k.

This analysis has implictly assumed only two-terminal nets. In the case of multiterminal nets with more than two terminals, it would appear that more than one crossbar might be required to implement both the lower level and higher level connections. With the flattened structure, this is not the case. Any net, with two or more terminals, can be routed in a single crossbar at the highest level that includes all of the FPGA's that contain the terminals of the net.

Rent's rule can be used to estimate the size of the crossbar at each level for systems with k levels. Rent's rule states that the number of wires leaving a subsystem that contains S components, each of which has P_B has pins, is $P = P_B \times S^R$, where P is the number of pins on the subsystem. P_B in this case is the number of pins on an FPGA. The Rent exponent R is typically in the range from 0.5 to 0.7. An exponent of 0.7 is assumed in the TM-2 interconnect to allow a high degree of routability.

Let m_k be the number of pins per FPGA leaving a level-k system, which contains 2^k FPGA's. The total number of pins across all FPGA's is predicted by Rent's rule to be $2^{k\times R}$, so m_k is $(2^{k\times R} \times P_B/2^k) = 2^{k\times(R-1)} \times P_B$ pins per FPGA. A level $k-1$ system has $m_{k-1} = 2^{(k-1)\times(R-1)} \times P_B$ pins per FPGA leaving level $k-1$. The difference between these two quantities, given in (1), is the number of pins that should be routed at level k

$$m_k - m_{k-1} = \left(2^{1-R} - 1\right) \times 2^{k(R-1)} \times P_B. \qquad (1)$$

This derivation predicts that a total of $2^N \times m_N$ pins should be available at the top level of the interconnect for use as I/O pins. This number is far in excess of those that are actually required for I/O—more than 2900 for our system. This is due to the fact that Rent's rule is only useful in predicting wire count for systems made up of a large number of components. To deal with this, we take the excess pins and distribute them across all levels, in proportion to the contribution of (1) to the total number of pins wired at all levels. Specifically, the total number of pins that should be connected at all levels from level 1 to N is given by

$$\sum_{i=1}^{N} \left(2^{1-R} - 1\right) \times 2^{i(R-1)} \times P_B. \qquad (2)$$

Distributing the total number of available pins according to the ratio of (1) to (2) in the total number of pins available, P_B, leads to (3) as the number of pins at each level

$$W_{k,N} = \frac{\left(2^{1-R} - 1\right) \times 2^{k(R-1)}}{\sum_{i=1}^{N} \left(2^{1-R} - 1\right) \times 2^{i}(R-1)} \times P_B. \qquad (3)$$

For a five-level system and 300 pin FPGA, this evaluates to $W_{k,5} = 107.3 \times 0.818^k$ wires at level k. In practice, the number of wires must be rounded up to a multiple of 2^k, since modularity requires that the same number of wires must go from each FPGA to each of the 2^k modules. Another practical consideration is that the wiring count at the top level is increased slightly to handle the I/O requirements. Fixing the I/O pins to specific physical pins on the TM-2 system means that there is no expectation of locality between an I/O pin on the TM-2, and the FPGA that connects to that pin.

TABLE I
I/O CAPACITY OF TM-2

System Size (# of FPGA's)	Total System I/O	Number of I/O modules	I/O per FPGA
2	64	1	32
4	128	2	32
8	256	4	32
16	256	4	16
32	512	8	16

TABLE II
ROUTING RESOURCES AT EACH LEVEL OF HIERARCHY

	System Size (# FPGA's)				
level	2	4	8	16	32
1	300	156	124	92	84
2		144	96	80	72
3			80	64	48
4				64	32
5					64

Fig. 4 shows the actual wire counts used at each level for a five-level system, rounding the value predicted by (3) up to a multiple of 2^k. This shows the crossbar for each level explicitly, in contrast to the single crossbar in Fig. 2(f). Table II contains similar data for all system sizes. For a five-level system, the $W_{k,5}$ from (3) are 87.1, 70.8, 57.5, 46.7, and 37.9 wires, which can be compared the actual counts shown in the last column of Table II. The disparity between (3) and actual counts is greatest for the five-level system due to the constraints on wire counts being a multiple of 2^k. This shows up most prominently at level 5 where the 37.9 wires predicted are rounded up to 64. Nevertheless, the counts are similar, except at the top levels where some of the level 3 and level 4 routing resources have been pushed into the level 5 routing. This is done to maximize the flexibility of the routing, and to provide support for the I/O network.

Fig. 5 illustrates the success of the nonuniform crossbar in reducing wiring count on the backplane. This plot shows the number of wires at each position on the backplane for both a uniform partial crossbar, and the nonuniform five-level one with the parameters listed above. It can be seen that the nonuniform partial crossbar offers a much lower, and relatively constant, wiring density.

It is also useful to define $w_{k,N} = (W_{k,N}/2^k)$, which is the number of wires per FPGA that go to each of the other FPGA's at level k. In a modular design, $w_{k,N}$ must be an integer, with the implication that $W_{k,N}$ must be a multiple of 2^k. At each level $k, 2^k - 1$ sets of $w_{k,N}$ wires go to the backplane and ultimately to crossbars on other modules, while $w_{k,N}$ wires are connected to the crossbar on this module. Another $2^k - 1$ sets of $w_{k,N}$ wires, which originate at FPGA's on other modules, are also connected to the crossbar. The total number of wires leaving a module is $\Sigma_{k=1}^{N}\ 2 \times (2^k - 1) \times w_{k,N}$.

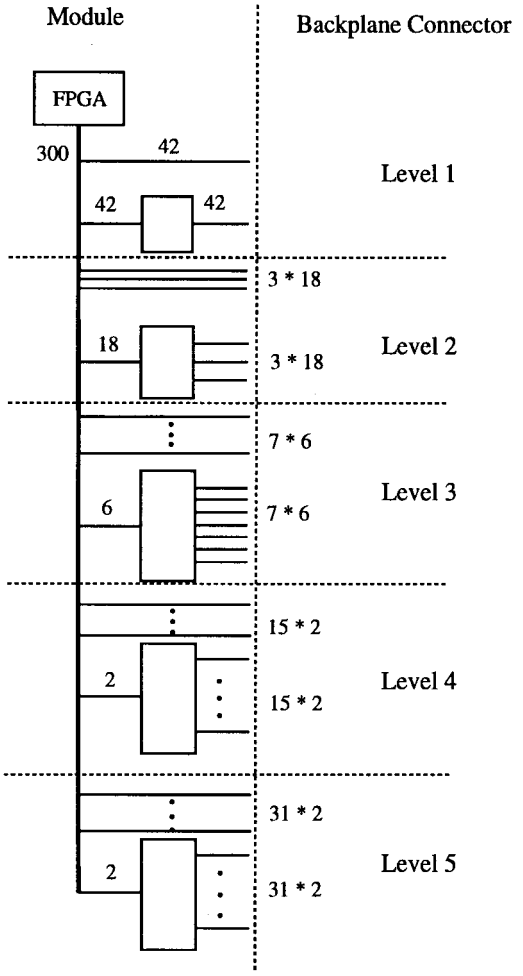

Fig. 4. Routing resource distribution for 32 FPGA system.

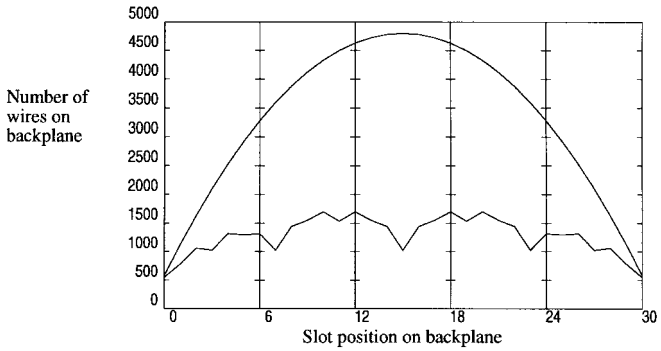

Fig. 5. Wiring density on backplane.

Table I indicates that the number of I/O's per module decreases with larger systems. This is inconvenient, as it makes the I/O connections differ between systems of various sizes. Instead, the TM-2 uses a uniform I/O count of 64 pins. Each module contains an I/O connector and a 64 × 64 crossbar to implement the outer stage of the routing network, but is only usable on a subset of the modules.

All discussions of the hierarchical interconnect up to this point have used a 32-module system as a design example. Table II gives the exact number of signals at each level for systems of various sizes, and introduces the complication

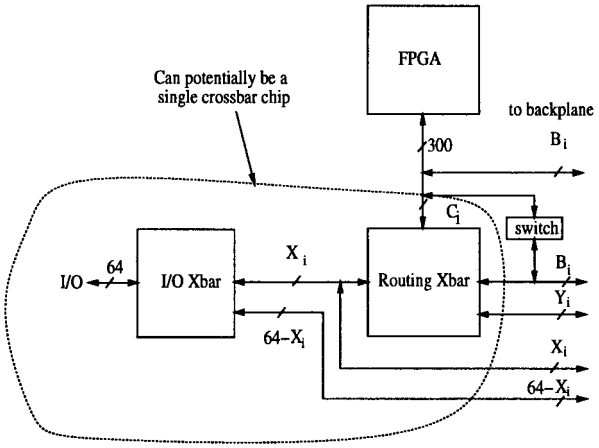

Fig. 6. Use of analog switches to reduce pin count on crossbar chip.

TABLE III
IMPLEMENTATION PARAMETERS FOR DIFFERENT SYSTEM SIZES

System Size	B_i	C_i	X_i	Y_i
2	150	150	32	32
4	186	114	16	48
8	204	96	8	56
16	222	78	4	60
32	230	70	2	62
max	230	150	32	62

that several different configurations of crossbars are required. Furthermore, for each system, a different number of signals go from the FPGA to the backplane interconnect, and from the FPGA to the crossbars on the module. Since each of the crossbars is relatively small (maximum 32 connections), it is possible to merge them into one or more larger crossbar chips. Fig. 6 illustrates this technique using a single physical crossbar to implement the FPGA routing, and a second crossbar to implement the outer stage of the routing network for the I/O. In fact, both of these can be merged into a single physical crossbar as shown. The pins B_i go to crossbars on other modules, the pins C_i go to the crossbar on this module. From the previous discussions, $B_i + C_i = 300$. In a system with N modules, each of the subset of modules in which the I/O connector is functional has an I/O connector and a 64×64 crossbar, with $X_i = (64/N)$ signals going to the routing crossbar, and $64 - X_i$ signals leaving the module for other modules. On modules without an I/O connector, 64 pins to the routing crossbar are used for routing the I/O signals from other modules.

Table III shows the value of these parameters for each size of system. In the most straightforward implementation, the crossbars are large enough to provide connections for the maximum number of signals used in any configuration. Pins that are excess to any configuration are left unused. The crossbar must have at least $128 + \max(B_i) + \max(C_i) + \max(X_i) + \max(Y_i) = 600$ pins. Pins used for B_i and C_i are the principal contributors to the total number of pins. The introduction of analog switches between these two sets of signals can allow the crossbar pins to be used as either B_i or C_i pins, as shown in Fig. 6. This changes the number of crossbar

pins required to $128+\max(B_i+C_i)+\max(X_i)+\max(Y_i) = 522$. It is not practical to employ this technique across all possible system configurations, but it does turn out to be useful to apply it selectively to reduce the number of crossbar chips required.

IV. Design of the TM-2

This section describes the practical issues associated with implementing the routing architecture, together with the methods used to meet the goals outlined in Section II of the paper.

A. Physical Constraints and Module Capacity

The practical limit on the number of PCB's that can fit on a single backplane is about 16. This immediately forces us to define a single PCB card as containing two modules. A module continues to be defined as having a single FPGA, RAM, and interconnect, but now two modules are placed on a single card, and the smallest system is two modules. This also slightly reduces the total number of wires on the backplane compared to an implementation with one module per PCB, as level 1 wiring is now contained within one PCB. Furthermore, the maximum wire length is half of the length that it would be with a 32-slot backplane. While the total number of wires on the backplane is slightly reduced, the number of pins per PCB is nearly doubled, requiring an 800 pin connector on the card.

B. RAM's

RAM access speed is likely to be the performance limit in many applications. Although all signals between FPGA's are routed through the interconnection network to maximize routing flexibility, the RAM's are directly connected to FPGA's to reduce delay. Two factors mitigate the loss of flexibility that this incurs. First, address and data pins are permutable in RAM's, so there is some freedom for the FPGA router to choose pins. Second, RAM's are less expensive than FPGA's, so it is reasonable to include a large amount of RAM, and provide the ability to use FPGA pins connected to unused RAM pins for general interconnect. As a result, certain FPGA pins are hardwired to RAM pins, and are also connected to the interconnect structure. Each FPGA on the TM-2 has two banks of RAM directly connected to the pins of the FPGA. Each bank, labeled "0" and "1," can contain up to 256K*64 bits. This requires a total of 174 address, data, and control pins on each FPGA. These pins are also connected to the lowest level interconnect that can provide a sufficient number of pins. The reason for choosing the lowest level interconnect for RAM pins is that higher level interconnect in the hierarchy can always be used for lower level purposes, but not vice versa. This requires the use of level 1, level 2, and level 3 interconnect, depending on the size of the system.

Use of the RAM's requires use of all pins connected to the associated bus. For example, using any part of any RAM requires using all 18 address pins and all associated control pins, even if only a fraction of the RAM is being used. Some flexibility is provided in that the RAM data bus can be configured as 8-, 16-, 32-, and 64-bit wide data paths, so narrow memories do not use the full 64 pins.

Direct control of the RAM's would require eight byte enables, read enable, and write enable pins for a total of 10 pins per RAM bank. FPGA pins are a relatively scarce commodity, and so to reduce the number of pins while still providing flexible control of the RAM's, each bank is controlled by a 5-bit code that encodes the size of the access, and provides the low address bits for accesses smaller than 64 bits.This method works well for systems that access memory as a linear array, but is poor for systems that want to access an arbitrary subset of bytes in a word, or substrings aligned at smaller boundaries, such as graphics applications. A better approach would have been to allow the choice of both encoded accesses and raw accesses.

C. Debugging Facilities

Debugging facilities must be provided so that the user can dynamically probe any signal in the system, after partitioning, placement, and routing of the system has been performed. Although debugging pins are similar to I/O, an important distinction is that it must be possible to select any signal for debugging without rerouting the entire system. It must be possible to access any signal while only performing incremental reconfiguration of the system. To support this, the TM-2 has a 32-bit bus that spans the entire system. Every crossbar chip is also connected to this debugging bus. Any signal that is available on an FPGA or I/O pin can be probed in real time by reconfiguring whichever crossbar chip is connected to it.

A second option for debugging is a JTAG serial chain. A JTAG chain is connected to all FPGA's and crossbar chips. Although this is much slower than the real-time debugging bus, it can inspect the state of the FPGA pins even when the pad drivers are disabled, which can happen during system overcurrent faults.

D. Module Design

Fig. 7 illustrates an overview of the design of the TM-2 PCB. The TM-2 uses I-Cube IQ320 Field-Programmable Interconnect Devices for the programmable interconnect. These devices implement a 320-pin crossbar, necessitating that the TM-2 interconnect be partitioned into several devices. The use of debugging connectors increases the total number of pins required slightly, but all of the interconnect for two TM-2 modules can be accommodated in four IQ320 devices. The I-Cube labeled **it** in the figure implements the outer stage of the routing network for the I/O signals, and the level 1 crossbar for both modules. Although the system was described in Section III as having one I/O connector per module, since no system has more than one I/O per two modules, a single I/O connector and crossbar is provided on each PCB. Connections between I-Cube **it** and I-Cubes **ibl** and **ibr** provide the signals between the two stages of the routing network. I-Cube **im** implements the crossbar at level-1 and higher for both modules, and I-Cubes **ibl** and **ibr** implement higher level crossbars for the left and right modules respectively. Some analog switches are required to share pins between the B_i and C_i pins for the 32-module configuration to implement all interconnect within four I-Cubes.

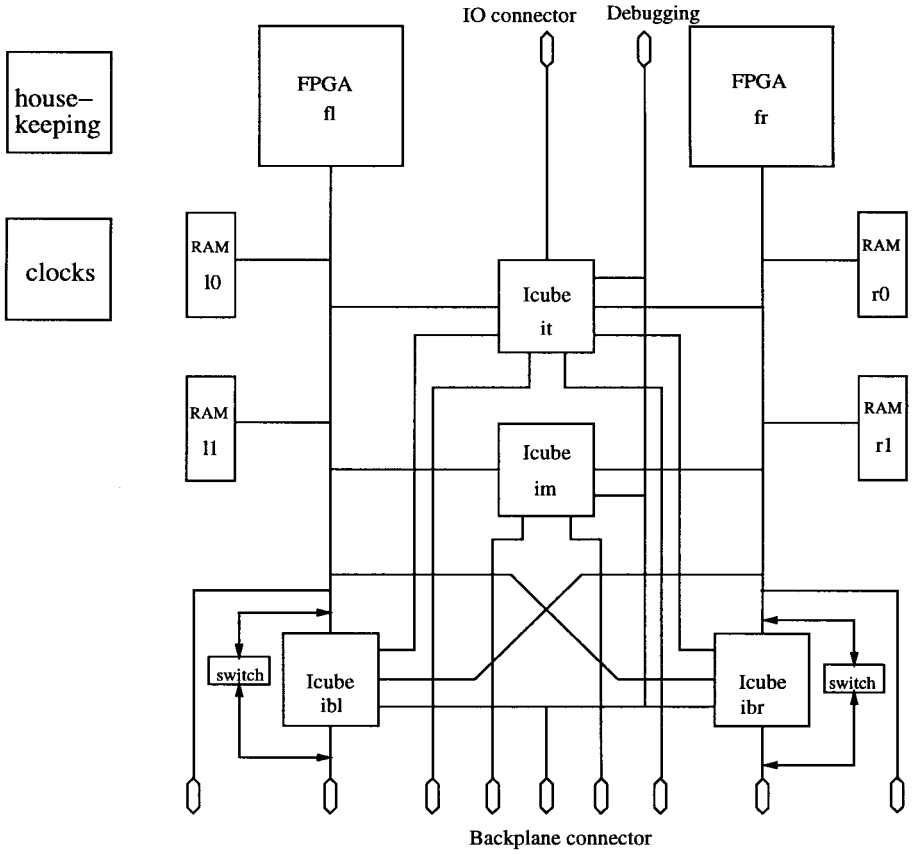

Fig. 7. Organization of the TM-2 printed circuit board.

E. Automated Design Using TM2-gen

Hand design of the TM-2 PCB would be a tedious and error-prone process. Instead, we implemented a program called **tm2_gen** to automate the design process. **Tm2_gen** is controlled by a small number of parameters that define the PCB. **Tm2_gen** allocates pins to the FPGA's and I-Cube crossbars according to the specification, and applies a number of consistency checks. It also generates the netlist for the PCB and for the backplane. Finally, it generates a system-level description file that specifies the connection of every FPGA to each of the I-Cube chips. This information is in the form of a flat file that makes no assumptions about the TM-2 hierarchical routing structure. It is used by the interchip global router **gr**, which reads in this description file and performs various consistency checks to make sure that the file describes a logically correct structure, and reports the total amount of available routing resources. **Gr** makes no assumptions about the routing structure, and has detected bugs in **tm2_gen** (unfortunately, after the first backplane was sent out for fab!)

F. Clocking

High-quality, low-skew programmable clocks are required by the user's circuits, as well as the ability to employ external clocks. The TM-2 has a field-configurable clock generator on each PCB, which generates four clocks, two of which are distributed on the clock networks of the Altera 10K50, and two of which are distributed on the low-skew routing networks of the Altera 10K50. The field-programmable clock generators are implemented in a clock FPGA, which generates 4-bit nibbles for each of the clock waveforms at a 25 MHz rate. Each bit represents the logic value of the clock for a 10-ns time interval, and the waveform pattern is serialized using a system-wide low-skew 100 MHz clock that is distributed by a high-quality clock tree. This enables user logic in a separate clock generation chip to produce clock patterns of arbitrary complexity with 10 ns edge resolution. A separate synchronization line is connected to all clock FPGA's to allow them to synchronize clock patterns. For example, a 70 ns clock would be generated using four distinct nibble patterns repeating in a periodic manner and the synchronization line would be used to signal the first of these.

Up to four external clocks can be used in the TM-2. To allow arbitrary phase related clocks to be generated, each of the external clocks enters a serial to parallel shift register, which provides nibble-wide data to the clock FPGA at a 25 MHz rate. The clock FPGA can then generate arbitrary clocks that are phase aligned to the external clocks.

Two more clock generators are provided for each RAM bank to control the Output Enable ($\overline{\text{OE}}$) and Write Enable ($\overline{\text{WE}}$) signals. Each of these clock generators is qualified by the logic that decodes the RAM control signals as described in Section IV-B. This allows the FPGA to generate control signals at its clock rate, while qualified write enable signals are generated to 10 ns resolution. Single-cycle RAM access is

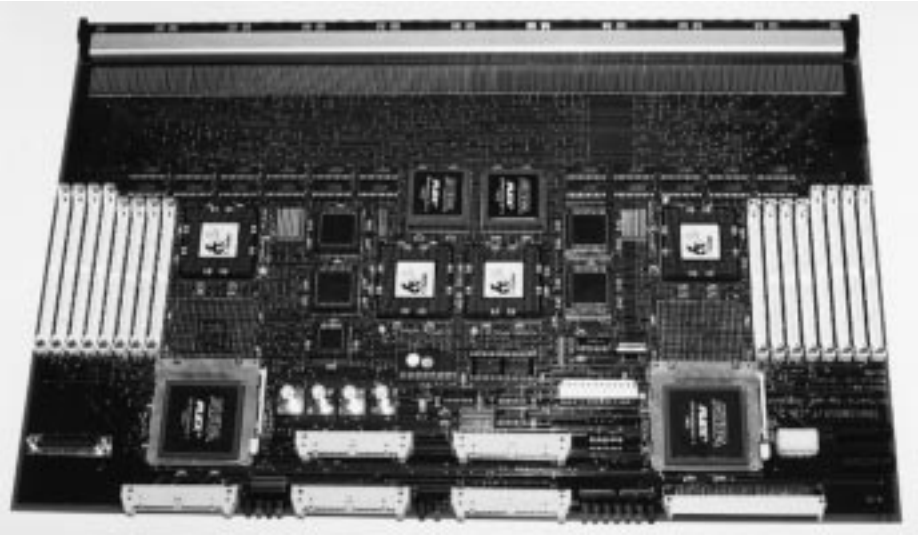

Fig. 8. The TM-2 printed circuit board.

possible with clock cycles only slightly longer than the RAM access time.

G. Status, Power Monitoring, Host Communication, and Boot

Each PCB contains a small FPGA, called the housekeeping chip, which is connected to a bus to allow configuration of the system and status monitoring. This bus is connected to a standard printer parallel port on a host SUN Sparcstation and uses a few wires on the TM-2 backplane. It allows each of the FPGA and interconnect chips in the system to be individually configured. A byte-serial protocol that allows rapid burst-mode transfers of a few hundred KB per second enables complete system configuration in a few seconds.

The housekeeping chip is also connected to each of the FPGA's via four wires called the nibble bus. A simple protocol is defined for this bus that can be used for moderate speed access to the user's circuit. High bandwidth host communication is performed over custom-designed hardware that attaches to the external I/O connectors.

Design errors may cause the user's circuit to drive conflicting logic values onto a net, causing an overcurrent into the FPGA's. Although the FPGA's are robust enough to tolerate a moderate level of abuse, it is desirable to detect this situation and shut down the system. Both the FPGA's and the I-Cube chips have dedicated pins that can disable the pad drivers. The TM-2 contains power supply current monitoring circuitry that will detect an overcurrent into the FPGA's or I-Cubes, shut down the appropriate chips, and provide an alert to the host.

V. Software

The CAD software for the TM-2 will ultimately provide a fully automated flow from the user's design to configuring the TM-2. This includes merging multiple design files, generated either from HDL's or schematics, partitioning the network across modules to fit the logic, embedded RAM's and external RAM's, global routing to provide connections between the modules, bit file generation, and interactive configuration and debugging. To date, we have completed only part of this flow. Designs at present must be manually partitioned among the FPGA's. Fully automated global routing is now in place, as well as a number of debugging and I/O features as described below.

A. Ports Package

The TM-2 port multiplexer package can be used to create a simple, easy-to-use, moderate speed communications path between a user's circuit and the host computer. Data can be sent between the circuit and a user's program running on a remote workstation over the network, with no hardware design required by the user. To use the package, the communication ports of the circuit are listed in a port description file that gives the name of each port, its direction (input or output), its width in bits and the names of any handshaking signals to be used. The ports package consists of two parts: a hardware generation package, and a software interface.

When the circuit is compiled for the TM-2, the hardware generation package synthesizes a wrapper circuit and adds it to the user's circuit. The wrapper circuit handles the details of transferring data to or from the ports of the user's circuit over the 4-bit nibble bus to the housekeeping chip, which will then communicate with a program running on the host (called **tm2mon**) over the parallel port.

The software side of the ports multiplexer package is a library of routines that can be called from a user program running on a workstation. Resembling the UNIX stdio package, the routines allow the program to open named ports on the circuit in the TM-2, and transfer data to or from them.

B. Global Router

The global router for the TM-2, called **gr**, determines the configuration of the I-Cube interconnect chips given the interchip connection netlist. **Gr** models the system as a collection of targets, each of which may have some hardwired resources directly connected to it, and a collection of crossbar chips. The targets are FPGA's, RAM's, I/O connectors, and LED's, and have an associated property describing pin permutability.

Gr reads an interconnect file, generated by **tm2_gen**, which describes the connections between targets and interconnect chip, and a netlist generated by the partitioner or user. **Gr** performs best-fit routing by sorting the nets according to their level of TM-2 routing required, and fanout, and uses the interconnect chip with the best fit to implement the net. **Gr** is written with simplicity as its foremost goal, and uses straightforward algorithms that are $O(n^2)$ in the number of pins. Despite this, it can route a full size system in under 20 s on a 70 MHz Sparcstation 5. **Gr** produces the configuration information for the I-Cube bit file generation, and a netlist information file that can be used to incrementally modify the routing for debugging purposes. It also determines the pin assignment for each signal that is required for each FPGA.

C. Tm2mon

Diagnostic and monitoring software for the TM-2 is built as a collection of small tools that communicate with a central control process, called **Tm2mon**. **Tm2mon** is a server process that is responsible for downloading, communication, status monitoring, and debugging the TM-2. It runs on the machine with the parallel port connection to the TM-2. It creates a local area network socket and accepts connections over which requests for downloading, status, and nibble bus protocol packets can be sent. This provides the highly useful ability to access the TM-2 hardware by other workstations on the network, including the Internet. Tools running on other machines interact with the TM-2 by communicating using the **tm2mon** protocol. In addition to standard download and status monitoring tools, users can create custom software front ends that interact with the TM-2. Debugging software can read the network routing file produced by **gr**, and incrementally modify the I-Cubes to access the desired signals on the debugging bus.

VI. Status and Plans

We have constructed and debugged two TM-2 PCB's. A picture of one is shown in Fig. 8.[1] We have constructed several simple circuits, and the compile and downloading process appears to be robust. Our most recent circuit is a triangle drawing circuit that rasterizes triangles into a frame buffer on a personal computer, transferring the data over a PCI bus. This comprises about 700 lines of C code, compiled using the tmcc compiler [9]. Other applications include DES key searching and procedural texture mapping. These designs can operate at 12–18 MHz, although an external interface limits the clock rate to 6.25 MHz.

We plan to perform minor hardware revisions and construct a full-scale 16 PCB system within the next year. This system will use the Altera 10K100 to implement over 2M gates. It will also incorporate a DRAM on each PCB to store multiple configurations of the FPGA's and I-Cubes. With this on-board configuration memory, a few dozen configurations can be stored on the TM-2, and downloaded in a few tens of milliseconds.

[1]For a color version, please see http://www.eecg.toronto.edu/~jayar/research/tm2.html.

VII. Conclusions

In this paper we have presented the architecture and design of the TM-2, a next generation field- programmable rapid-prototyping system. The major feature of the architecture is the novel routing architecture, which avoids the high wiring density that would be incurred by a regular crossbar structure.

At present, two PCB's have been built and tested. One PCB has been used to prototype several graphics acceleration hardware algorithms, which directly drive the frame buffer of an personal computer through a PCI bus interface. A large effort has been placed into the software system which automatically routes and programs the complete system.

Acknowledgment

The authors wish to acknowledge the donations of components from Altera and I-Cube.

References

[1] M. Butts, J. Batcheller, and J. Varghese, "An efficient logic emulation system," in *Proc. ICCD*, 1992, pp. 138–141.
[2] M. Slimane-Kadi, D. Brasen, and G. Saucier, "A fast-FPGA prototyping system that uses inexpensive high-performance FPIC," in *Proc. 2nd Annu. Workshop FPGAs*, 1994.
[3] R. Tessier, J. Babb, M. Dashl, S. Hanon, and A. Agarwal, "The virtual wires emulation system: A gate-efficient ASIC prototyping environment," in *Proc. 2nd Annu. Workshop FPGAs*, 1994.
[4] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable active memories: Reconfigurable systems come of age," in *Proc. IEEE Trans VLSI Syst.*, vol. 4, pp. 56–69, Mar. 1996.
[5] J. Arnold, D. Buell, and E. Davis, "Splash 2," in *Proc. 4th Annu. ACM Symp. Parallel Algorithms Arch.*, 1992, pp. 316–322.
[6] R. Amerson, R. Carter, W. Culbertson, P. Keukes, and G. Snider, "Teremac—Configurable custom computing," in *Proc. FPGA-95*, 1995, pp. 32–29.
[7] D. Galloway, D. Karchmer, D. Chow, D. Lewis, and J. Rose, "The transmogrifier: The University of Toronto Field-Programmable System," in *Proc. Second Canadian Workshop Field-Programmable Devices*, Kingston, Ont., Canada, June 1994. Also available as CSRI Tech. Rep. 306 via anonymous ftp from ftp://ftp.csri.toronto.edu/csri-technical-reports/306/.
[8] P. Chan and M. Schlag, "Architectural tradeoffs in programmable-device-based computing systems," in *Proc. FPGA's for Custom Computing Machines*, 1993, pp. 152–161.
[9] D. Galloway, "The transmogrifier C hardware description language and compiler for FPGA's," in *Proc. IEEE Symp. FPGA's Custom Comput. Machines FCCM '95*, Apr. 1995.
[10] M. Khalid and J. Rose, "The effect of fixed I/O pin positioning on the routability and speed of FPGA's," in *Proc. Canadian Workshop Field-Programmable Devices FPD 95*, pp. 94–102.
[11] *Altera Flex Logic Handbook*, Altera Corp., 1996.

David M. Lewis (M'86) received the B.A.Sc. degree with honors in engineering science and the Ph.D. degree in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1977 and 1985, respectively.

From 1982 to 1985, he was employed as a Research Associate on the Hubnet project and developed custom integrated circuits for a 50-Mb/s local area network. He became an Assistant Professor at the University of Toronto in 1985, and since 1991, he has been a Associate Professor. His research interests include logic and circuit simulation, logarithmic arithmetic, signal processing, field programmable gate arrays and systems, and VLSI architecture.

David R. Galloway (S'78–M'80) received the B.A.Sc. degree in engineering science and the M.Sc. degree in computer science from the University of Toronto, Toronto, Ont., Canada, in 1977 and 1979, respectively.

From 1979 to 1996, he worked for the Computer Systems Research Group, University of Toronto as a Unix Systems Programmer, Systems Manager, and Research Associate. Since 1992, he has worked part time for the Department of Electrical and Computer Engineering, University of Toronto on computer-aided design for field-programmable systems. He is currently an independent consultant, specializing in FPGA CAD and Internet-based systems for foreign currency conversion.

Marcus van Ierssel received the B.A.Sc. degree in engineering science and the M.Sc. degree in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1992 and 1995, respectively.

Since 1995, he has been working for the Department of Electrical and Computer Engineering at University of Toronto as a Research Associate. He has been working on the design and construction of the Transmogrifier-2 and a set-top box internet browser.

Jonathan Rose (S'79–M'80) received the B.A.Sc. degree in engineering science in 1980 and the M.A.Sc. and Ph.D. degrees in electrical engineering in 1982 and 1986, respectively, from the University of Toronto, Toronto, Ont., Canada.

From 1986 to 1989, he was a Research Associate in the Computer Systems Laboratory at Stanford University, Stanford, CA. In 1989, he joined the Faculty of the University of Toronto, where he is currently an Associate Professor of Electrical and Computer Engineering. He spent the 1995–1996 year as a Senior Research Scientist at Xilinx, Inc., San Jose, CA, working on a next-generation FPGA architecture. He is the cofounder of the ACM FPGA Symposium, and remains part of that Symposium on its steering and program committees. His research covers all aspects of FPGA's including computer-aided design (CAD) and architecture for FPGA's, field programmable systems, and applications of rapid prototyping sytems.

Paul Chow (S'79–M'83) received the B.A.Sc. degree with honors in engineering science and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ont., Canada, in 1977, 1979, and 1984, respectively.

In 1984, he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Research Associate, where he was a major contributor to the MIPS-X project. In January 1988, he joined the Department of Electrical and Computer Engineering at the University of Toronto, where he is now an Associate Professor. His research interests include high-performance computer architectures, architectures for programmable digital signal processors, VLSI systems design, and field programmable gate array architectures and applications.

Dr. Chow currently serves on the Technical Advisory Committee for the Canadian Microelectronics Corporation.