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

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

Department of Electrical and Computer Engineering

University of Toronto

Abstract

This paper describes the Transmogrifier-2, a second generation multi-FPGA system. The largest version of the system will comprise 16 boards that each contain two Altera 10K50 FPGAs, 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 non-uniform partial crossbar, that provides a constant delay between any two FPGAs in the system. The TM-2 architecture is modular and scalable, meaning that various sized systems can be constructed from 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 resolution to 10ns.

The first Transmogrifier-2 boards have been manufactured and are functional. They have recently been used successfully in some simple graphics acceleration applications.

1 Introduction

Continuing advances in the density and speed of FPGAs have made them effective implementation vehicles for increasingly complex systems. Nevertheless, contemporary FPGAs lag semi-custom ASICs by a factor of 10 or more in density, and lack system-level facilities, such as very large RAM and clocks, 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 moderate number of field-programmable systems have been described [1] [2] [3] [4] [5] [6] [7], and can be roughly classified as emulation systems, or custom compute engines. Emulation systems tend to be targeted at a wide range of applications, but suffer from low clock rates, while custom compute engines tend to have architectures that constrain the applications to those that fit the computational model offered by the hardware. The primary motivation for the Transmogrifier-2 (or TM-2, for short) is a flexible rapid prototyping system that offers high capacity, high clock rates, and is flexible enough to implement a wide variety of systems.

We have previously constructed a small-scale rapid prototyping system, the Transmogrifier-1[7]. It contained only 40K FPGA gates and 128KB RAM, and was only capable of implementing small systems. Further, constructed from a standard 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 2 describes the goals of this project. Section 3 describes the routing architecture developed for the TM-2. In Section 4, the influence of the goals and technical constraints on the detailed design are described. The software development system for the TM-2 is described in Section 5, and Section 6 concludes with a report of the project's current status.

2 Goals

Our experience with implementing applications on the Transmogrifier-1, and the desire to create an effective, large-scale and easily usable system lead to the following goals for the Transmogrifier-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 capable of being used 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 single FPGA together with sufficient routing resources to provide all interconnect to the rest of the system.
2. **1 Million Gate Capacity, modern FPGAs**. The system should be configurable up to 1M gates plus RAM, and use modern FPGAs to achieve this capacity. The TM-2 uses the Altera 10K50, with 35K usable logic gates (not including embedded RAM). A 32-FPGA system exceeds 1M gates.
3. **Well-behaved Inter-Chip 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 10Mhz when the user is unaware of the underlying architecture of the system. 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 necessary to generate the write enable signal for a RAM, which must be precisely controlled but is 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 FPGAs and total system cost, we prefer to provide an excess amount of RAM, much of which is not 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 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 re-compiling 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 seconds.
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.

3 Routing Architecture

The inter-chip interconnect architecture is the key aspect of the TM-2 that determines system performance. We make the following assumptions concerning the qualities of the FPGAs, manufacturability concerns and usage of the system by users:

- We assume that the pin assignment on the FPGAs can be imposed by the software system - i.e. that both routability and speed of the individual FPGAs is maintained in the presence of arbitrary pin permutations, as studied in [10].
- System I/O signals are assigned to specific pins as specified by the user.
- A single module contains both the FPGA on a board, and any programmable interconnect to support the FPGAs on the board 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 assume that the FPGA has 300 pins, although the Altera 10K50 actually has 304 usable 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 FPGAs, each of which has 300 signal pins would require $32 \times 300 = 9600$ wires from the FPGAs 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 Clos 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 FPGAs, provided that the FPGAs 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 outer stages are 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 FPGAs 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 FPGAs 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. TM-2.

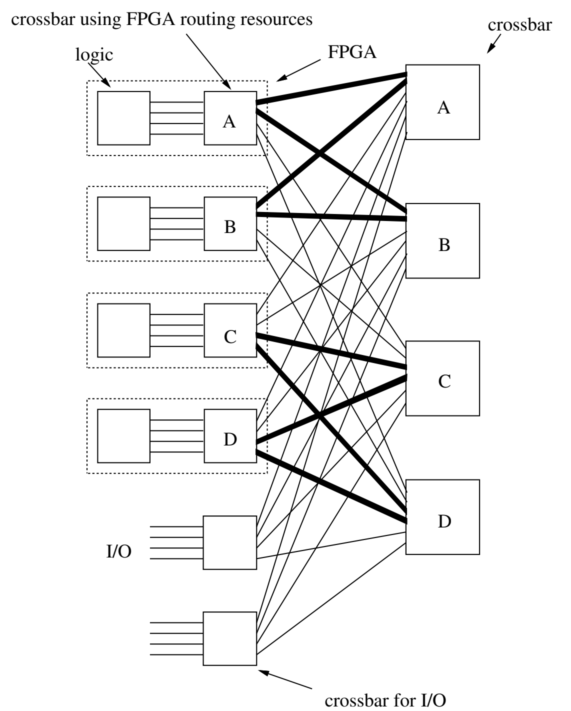
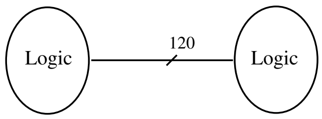
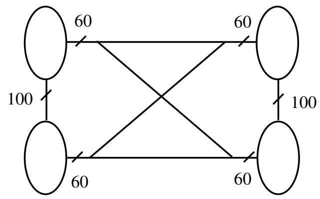


Figure 3.1 Partial Crossbar with Non-uniform Routing Density

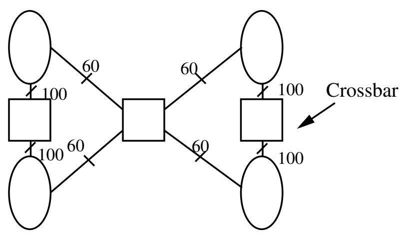
The TM-2 routing structure uses the partial crossbar structure, but modified to take advantage of the natural hierarchy and resulting locality of wiring within circuits. Figure 3.1 shows a partial crossbar architecture, containing four FPGAs and two I/O modules. The internal routing of the FPGA provides one stage of crossbar, and separate routing chips interconnect the FPGAs. 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. 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 Figure 3.1, the circuit has a higher degree of local connectivity between FPGAs 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 FPGAs is reflected in the different number of wires between these FPGAs and physically nearby crossbars. Although the local wiring could be distributed across all of the crossbars, concentrating it in physically nearby crossbars reduces the maximum wiring density of backplane, although the total number of wires is constant.



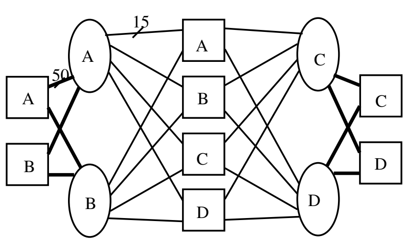
(a) Bipartitioning of Circuit



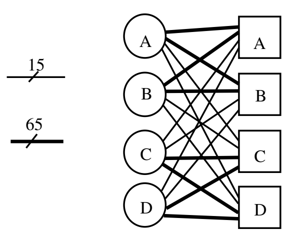
(b) Second Bipartitioning of Circuit



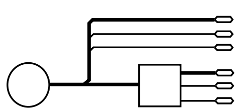
(c) Implementation of Circuit Using Crossbars



(d) Splitting Crossbars into Partial Crossbars



(e) Merging Partial Crossbars for Multiple Levels



(f) Structure of a Single Module

Figure 3.2 Construction of Non-uniform Partial Crossbar using Bipartitioning

3.1 Relation of WIre Counts to Hierarchical Bipartitioning

This section shows how the wire counts 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 the we are given some circuit, and wish to implement it in four FPGAs, and will recursively bipartition the network to fit in the FPGAs. Figure 3.2 illustrates how the wiring can be distributed across the crossbars in a manner corresponding to the hierarchical partitioning of a circuit. Assume that we are given some circuit, and bipartition it. Figure 3.2(a) the first bipartitioning of the network results in 120 signals that cross the top cut. In Figure 3.2 (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 them. Figure 3.3(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 in Figure 3.2(d). Next, in Figure 3.2(e) these are grouped together so that each FPGA has an associated crossbar (although the physical implementation might be more than a single chip.) Finally, Figure 3.2(e) shows a modular structure which contains a single FPGA and 160 pin crossbar (65+65+15+15) implementing the two separate partial crossbars. Backplane wiring according to the density in Figure 3.2(f) is used to connect the modules. The maximum backplane wiring density is 190 $(65 \times 2 + 15 \times 4)$ between modules A and B, or C and D, in contrast to the 320 wires that would be required for a uniform partial crossbar.

3.2 Derivation of Crossbar Sizes

The next step is to determine the wiring requirements for a non-uniform partial crossbar according to the routing requirements of a recursively bipartitioned network. This can be done by viewing the circuit as a hierarchical network, and estimating the number of signals that exist at each level in the network. As shown above, the number of wires that are local to each level of the hierarchy correspond to wires that are induced by cutting the network at that level. 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 network is bipartitioned.

A circuit in the TM-2 can be viewed as a hierarchy of networks, with a level-k system comprising 2^k FPGAs. A level $k+1$ system is constructed with 2 level-k systems and some programmable interconnect. Conceptually, a level-$k+1$ system is constructed by using a level-$k+1$ crossbar to interconnect two level-k systems. In fact, the technique demonstrated in Figure 3.2 is used to implement all levels of the hierarchy at a single level of routing. Figure 3.3 illustrates the relationship between a hierarchical routing architecture and our routing architecture. Figure 3.3(a) illustrates a hierarchical routing architecture. Figure 3.3(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 $m_{k+1} - m_k$ at level k.

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 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, in this case 300. The Rent exponent R is typically in the range of 0.5 to 0.7. An exponent of 0.7 is assumed in this work to allow the a high degree of routability.

Let m_k be the number of pins per FPGA leaving a level-k system,

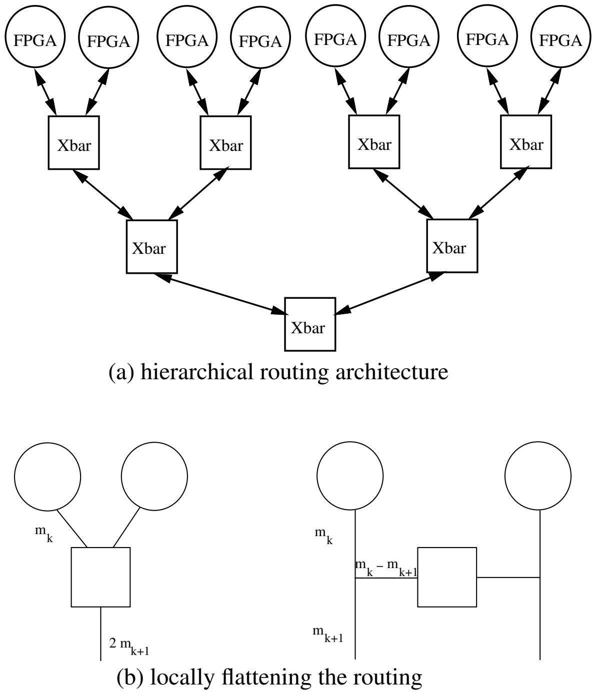


Figure 3.3 Flattening a Hierarchical Routing Architecture

which contains 2^k FPGAs. The total number of pins across all FPGAs is predicted by Rent's rule to be $2^{k \times R} \times P_B$, so m_k is $2^{k \times (R-1)} \times P_B$ pins per FPGA. A level $k+1$ system has $2^{(k+1) \times (R-1)} \times P_B$ pins per FPGA leaving level $k+1$. The difference between these two quantities, $(1 - 2^{(R-1)}) \times 2^{k(R-1)} \times P_B$, is the number of pins that should be associated with the level $k+1$ crossbar. For $R = 0.7$ and $P_B = 300$, the number of FPGA pins connected at level $k+1$ is $0.188 \times 0.812^k \times 300 = 56 \times 0.812^k$.

Figure 3.4 shows the details of the routing predicted by this equation. At each level k, $2^k - 1$ sets of w wires go to the backplane and ultimately to crossbars on other modules, while w wires are connected to the crossbar on this module. Another $2^k - 1$ sets of w wires, which originate at FPGAs on other modules, are also connected to the crossbar. The use of partial crossbars imposes the constraint that the number of connections from an FPGA to the crossbar at level k must be an integer multiple of 2^k. The number of connections at each level in Figure 3.4 has been slightly adjusted to meet this constraint.

The preceding discussion has ignored the signals which fall off the bottom of the system, and according to Rent's rule, would be used for I/O from the entire system. I/O requirements are actually lower than those suggested by the number of pins at the bottom, due to the fact that Rent's rule only applies to sub-systems smaller than 1/5 the total system size. This can be dealt with by choosing an appropriate number of I/O pins for expected system requirements, shown in Table 3.1, and linearly scaling up the size of the various crossbars in proportion to the increased number of interconnect signals. For example, in the largest system of size 32, we allow 512 I/Os, or 16 I/O per FPGA. The routing switch requirements as determined by Rent's rule specify a total of 92 I/Os per FPGA. We adapt the

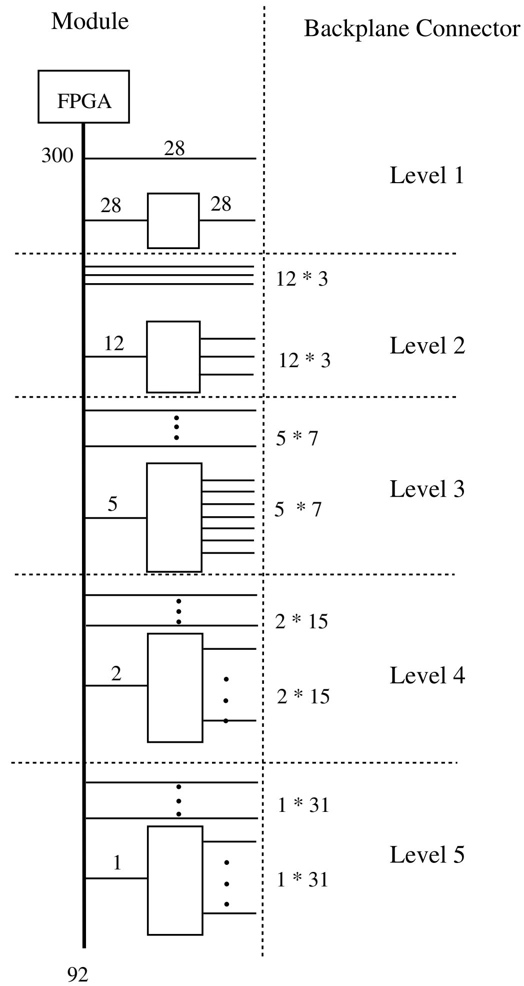


Figure 3.4 Routing Resource Distribution for 32 FPGA System

architecture to the smaller number of I/O pins required by adding an extra 16 signals from each FPGA to the top level routing crossbar, and including a corresponding number of outer crossbars to connect the I/O connectors to the inner stage routing network. This leaves $92 - 16 = 76$ FPGA pins with no connection, while 208 FPGA pins are connected to the routing network. These extra 76 pins are allocated to the routing network in the same proportion as the first 208, so the number of pins at each level is scaled up by the factor $(1 + \frac{76}{208})$. Because the number of pins at each level must be a multiple of 2^k, the number of pins is adjusted up to the nearest integer multiple of 2^k. Table 3.2 shows the actual number of pins required at each level for systems of various sizes. A consequence of this scaling process is that the number of pins at each level is now dependent on the size of the system, whereas previously it was a fixed value at each level. This must be accommodated by allowing pins to be used for routing at more than a single level, depending on system size. The method used to support this will be described shortly.

These requirements are dealt with by linearly scaling up the interconnect counts at each level to distribute the excess I/Os across all levels of interconnect, and providing a logically separate crossbar for the outer stage of the routing network. Because each of the crossbars in Figure 3.4 are small, they can be merged into a single physical crossbar chip, increasing pin permutability as a side

Table 3.1 I/O Capacity of TM-2

System Size (# of FPGAs)	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 3.2 Routing Resources at Each Level of Hierarchy

	System Size (# FPGAs)				
routing 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

effect.

Table 3.1 indicates that the number of I/Os 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 chunk 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 3.2 gives the exact number of signals at each level for systems of various sizes, and introduces the complication 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 a one or more larger crossbar chips, which implement the set of smaller crossbars. Figure 3.5 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, $B_i + C_i = 300$. On modules that have an I/O connector a 64×64 crossbar is implemented, with, $X_i = \frac{64}{N}$ 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

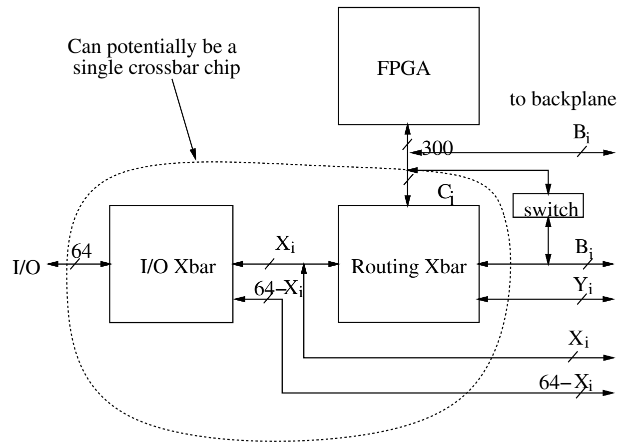


Figure 3.5 Use of Analog Switches to Reduce Pin Count on Crossbar Chip

modules.

Table 3.3 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 Figure 3.5. This changes the number of crossbar pins required to $128 + max(B_i + C_i) + max(X_i) + max(Y_i) = 520$. 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.

Table 3.3 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	60

4 Design of The TM-2

This section describes the practical issues associated with implementing the routing architecture, together with the methods

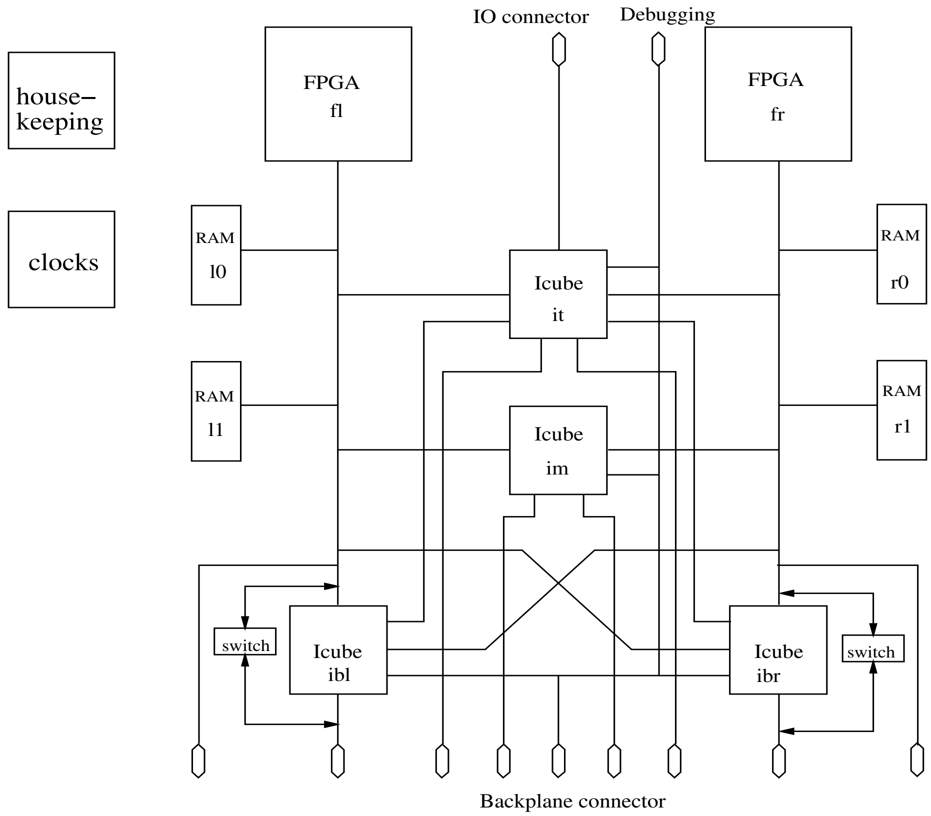


Figure 4.1 Organization of the TM-2 Printed Circuit Board

used to meet the goals outlined in Section 2 of the paper.

4.1 Physical Constraints and Module Capacity

The practical limit on the number of boards that can fit on a single backplane is about 16. This immediately forces us to define a single PC 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.

4.2 RAMs

RAM access speed is likely to provide one of the performance limits in many applications. Although all signals between FPGAs are routed through the interconnection network in order to maximize routing flexibility, the RAMs are directly connected to FPGAs to reduce delay. Two factors mitigate the loss of flexibility that this incurs. First, address and data pins are permutable in RAMs, so there is some freedom for the FPGA router to choose pins. Second, RAMs are less expensive than FPGAs, so it is reasonable to include an excessive 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.

The TM-2 has two banks of RAM, connected directly to each FPGA. Each bank, labelled 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.

Direct control of the RAMs would require 8 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 RAMs, 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 might have been to allow both encoded access and optionally raw accesses, at the cost of using more FPGA pins.

4.3 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 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 FPGAs and crossbar chips. Although this is much slower than the real-time debugging bus, it can inspect the state of the FPGAs even when the pad drivers are disabled, which can happen during system overcurrent faults.

4.4 Module Design

Figure 3.5 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 labelled 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 3 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 it and ibl and ibr provide the signals between the two stages of the routing network. Icube im implements the crossbar at level-1 and higher for both modules, and 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 Icubes.

4.5 Automated Design Using TM2-gen

Hand design of the TM-2 board 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 board. **Tm2_gen** allocates pins to the FPGAs and I-Cube crossbars according to the specification, and applies a number of consistency checks. It also generates the netlist for the board and for the backplane. Finally, it generates a system level description file that specifies the connection of every FPGA to each of the Icube 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 inter-chip 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** has no assumptions about the routing structure, and has detected bugs in **tm2_gen** (unfortunately, after the backplane was sent out for fab!)

4.6 Clocking

High quality, low-skew programmable clocks are required for use by the user's circuits, as well as the ability to employ external clocks. The TM-2 supports four clocks per FPGA, 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. Each of these clocks is generated by a programmable clock generator. The field-programmable clock generators are implemented in a clock FPGA, which generates 4-bit nibbles for each of the clock waveforms at a 25Mhz rate. Each waveform pattern is serialized using a 100Mhz 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 10ns edge resolution. A separate synchronization line is connected to all clock FPGAs to allow them to synchronize clock patterns.

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 25Mhz 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 4.2. This allows the FPGA to generate control signals at its clock rate, while qualified write enable signals are generated to 10ns resolution. Single cycle RAM access is possible with clock cycles only slightly longer than RAM access time.

4.7 Status, Power Monitoring, Host Communication and Boot

A small FPGA, called the housekeeping chip, is connected to a bus to allow configuration of the system and status monitoring. It connects to a standard printer parallel port on a host SUN Sparcstation and uses a minimum number of wires on the 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 addition of a bank of RAM to the housekeeping chip is being considered for the second revision of this board. This would store up to 10 configurations per DRAM chip, and allow reconfiguration in less than 100ms.

The housekeeping chip also is connected to each of the FPGAs 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 FPGAs. Although the FPGAs are robust enough to tolerate a moderate level of abuse, it is desirable to detect this situation and shut down the system. Both the FPGAs 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 FPGAs or I-Cubes, shut down the appropriate chip, and provide an alert to the host.

5 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 HDLs or schematics, partitioning the network across modules to fit the logic, embedded RAMs and external RAMs, 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 FPGAs. Fully automated global routing is now in place, as well as a number of debugging and I/O features as described below.

5.1 Ports Package

The TM-2 port multiplexor package can be used to create a simple, easy to use, low data rate 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 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 circuit, which will then communicate with a program running on the host (called **tm2mon**) over the parallel

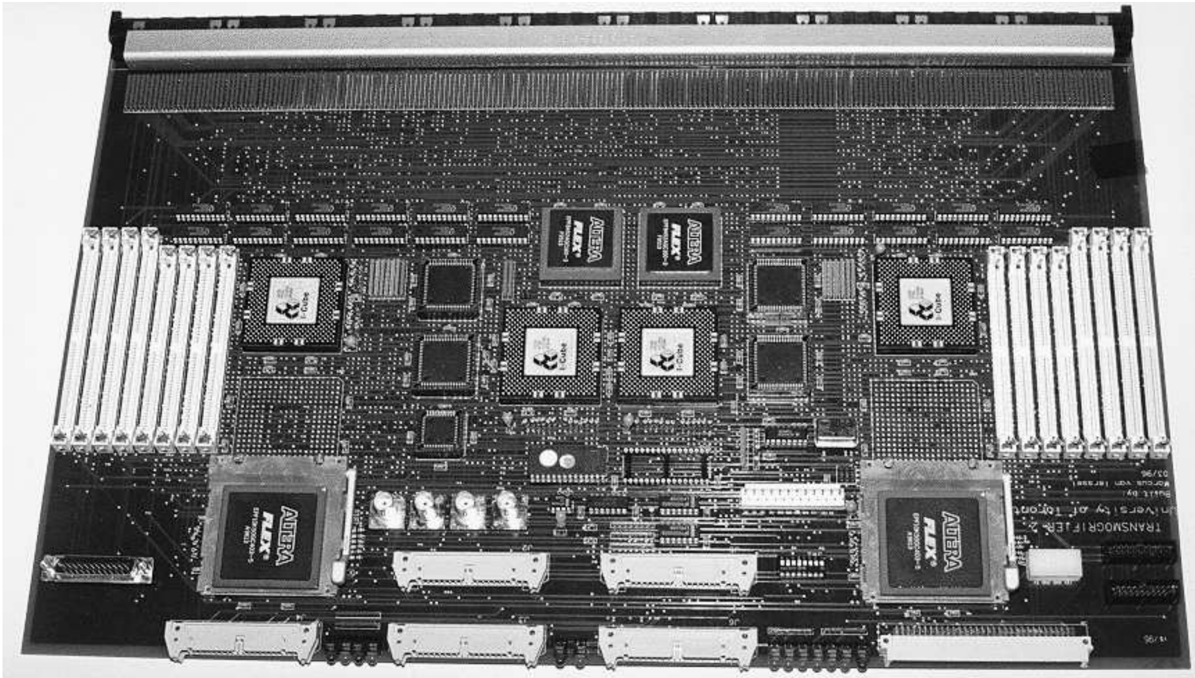


Figure 6.1 The TM-2 Printed Circuit Board

port.

The software side of this interface port multiplexor 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.

5.2 Global Router

The global router for the TM-2, called **gr**, determines the configuration of the interconnect chips given the inter-chip 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 FPGAs, RAMs, I/O connectors, and LEDs, 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 where $O(n)$ algorithms could be written. Despite this, it can route a full size system in under 20 seconds on a 75Mhz 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 positioning that will be required of the FPGAs.

5.3 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 feature that the base hardware can be easily accessed and manipulated 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.

6 Status

We have constructed and debugged two TM-2 boards. A picture of one is shown in Figure 6.1. For a colour version, please see http://www.eecg.toronto.edu/~jayar/research/tm2.html. 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 over a PCI bus and is displayed on a personal computer. This comprises about 700 lines of C code, compiled using the **tmcc** compiler [9]. We plan to perform minor hardware revisions and construct a full 1-M gate system within the next year.

7 Conclusions

In this paper we have presented the architecture and design of the Transmogrifier-2, a next generation Field-Programmable rapid prototyping system. The major feature of the architecture is its routability and the constant delay between any two chips in the system.

The status of the project is that two modules (boards) have been built and tested. One board 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.

8 Acknowledgments

The authors wish to acknowledge the donations of components from Altera and I-Cube, and the direct funding from NORTEL Technologies, the Natural Sciences and Engineering Research Council of Canada, and Ricoh Corporation.

9 References

[1] M. Butts, J. Batcheller, J. Varghese, "An Efficient Logic Emulation System", Proc. ICCD, 1992, pp. 138-141.

[2] M. Slimane-Kadi, D. Brasen, G. Saucier, "A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC", in Proc. 2nd Annual Workshop on FPGAs, 1994

[3] R. Tessier, J. Babb, M. Dashl, S. Hanon, A. Agarwal, "The Virtual Wires Emulation System: A Gate-Efficient ASIC Prototyping Environment", Proc.2nd Annual Workshop on FPGAs, 1994

[4] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age", Proc. IEEE Trans VLSI Systems, March 1996, pp. 56-69

[5] J. Arnold, D. Buell, E. Davis, "Splash 2", in Proc. 4th Annual ACM Symp. on Parallel Algorithms and Architectures, pp 316-322, 1992

[6] R. Amerson, R. Carter, W. Culbertson, P. Keukes, G. Snider, "Teremac - Configurable Custom Computing", in Proc. FPGA-95, pp 32-29

[7] D. Galloway, D. Karchmer, D. Chow, D. Lewis, J. Rose, "The Transmogrifier: The University of Toronto Field-Programmable System", Second Canadian Workshop on Field-Programmable Devices, Kingston, Ontario, June 1994. Also available as CSRI Technical Report 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. FPGAs for Custom Computing Machines, 1993, pp 152-161

[9] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs", IEEE Symposium on FPGAs for Custom Computing Machines, FCCM '95, April 1995

[10] M. Khalid and J. Rose, "The Effect of Fixed I/O Pin Positioning on The Routability and Speed of FPGAs," Proc. Canadian Workshop of Field-Programmable Devices, FPD 95, pp. 94-102.