Wormhole Run-time Reconfiguration

Ray Bittner and Peter Athanas

Virginia Polytechnic Institute and State University
The Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia 24061-0111

ABSTRACT

Configurable Computing Machines (CCMs) are an emerging class of computing platform which provide the computational performance benefits of ASICs, yet retain the flexibility and rapid reconfigurability of general purpose microprocessors. In these platforms, computational "hardware" is essentially swapped in and out of the platform as needed, much like paging in virtual memory systems. For an efficient platform, the swapping of the computational hardware (referred to as Run-Time Reconfiguration, or RTR) must be rapid. Thus far, the means of altering the configuration of CCMs has relied on global control strategies that present a fundamental bottleneck to the potential bandwidth of configuration information flowing into the CCM. Wormhole Run-time Reconfiguration is presented as a distributed control methodology that is applicable not only to the problem of device-level CCM reconfiguration, but to system-wide concurrent computing as a whole. The Virginia Tech Colt/Stallion integrated circuits are computational FPGAs incorporating Wormhole RTR concepts, and are discussed as a case study.

Keywords: Configurable computing, FPGA, digital signal processing, data flow, VLSI

1. INTRODUCTION

Configurable Computing Machines (CCMs) achieve high computational performance by creating custom data paths, operators, and interconnection pathways for a given problem. Run-time reconfiguration (RTR) is an implementation approach which divides an application into a series of sequentially executed stages, with each stage implemented as a separate execution module [1]. Partial RTR extends this approach by partitioning these stages into finer-grain sub-modules which are constructed to be swapped into the platform as needed to contribute towards a given computation. In such fashion, new dimensions and processing capabilities are added to configurable computing machinery, which would otherwise simply function as "ASIC emulators." Because of its unique computing paradigm, RTR offers an opportunity to re-examine many of the organizational assumptions of programming and computation. Unlike conventional computers, the functionality and organization of RTR systems are allowed to change dynamically as a function of the data.

Contemporary CCMs rely upon RAM-based FPGAs as the mechanism of achieving reconfigurability. Even though these devices were not designed for computing, they have proven quite effective in demonstrating the potential for high-performance computing. One of the limitations of contemporary FPGAs is the reconfiguration mechanism. There are two primary approaches commonly adopted: serial configuration and random access configuration. In devices using serial configuration, such as the Xilinx 4000 series [2] and the Altera FLEX 8000 line [3], the configuration storage elements are connected as a large scan chain around the entire chip. During configuration, the programming information is loaded into the device and shifted throughout in a bit-wise manner. While parallel loading of the configuration data is possible with some devices, close examination of the timing employed by these devices reveals an internally serial architecture. Most often, the entire chip must be programmed in such fashion before any part of it may be used to perform useful computation.

In a run-time reconfigurable environment, the serial method of configuration has a few disadvantages. For one, the concept of partial reconfiguration is not supported. There are many occasions in configurable computing when it is advantageous to reconfigure part of a device while leaving the rest intact. Changing the constant used in a particular calculation is one good example. Also, the configuration/execute cycle must be done sequentially due to the fact that the entire device must be configured before any part may be used for execution. In a run-time reconfigurable machine, this configuration/execute cycle may be compared to the Von Neumann fetch/execute cycle of a microprocessor. All high speed microprocessors of today overlap the fetch and execute cycles to some degree in an attempt to gain the speedups offered by a more pipelined approach. It would seem advantageous to barrow this concept and apply it to CCMs. Moreover, the scheduling of multiple independent processes onto a single device is also hampered by an all-or-nothing configuration scheme. The ability to configure one region of a chip while simultaneously executing within another region would aid not only in pipelining the configuration/execution cycle for a single process, but also in the multitasking of several processes onto the same chip. This concept will become increasingly valuable as the size and density of FPGA devices increases. Further, as more full-sized CCM systems are developed consisting of multiple devices, the ability to share the same set of resources across multiple processes, or different threads of the same process, will prove it's worth.

Another deficiency of the serial configuration method is a lack of speed. The benefits gained from the use of a set of truly parallel configuration lines is intuitively obvious. In CCMs, speed of configuration often translates directly into performance, particularly when multiple configurations must be swapped in and out of a device in quick succession. Thus, it

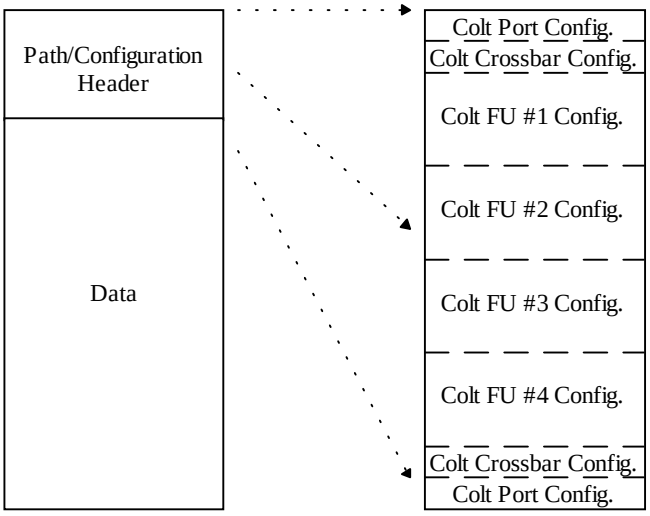

Figure 1 - An illustration of the stream format. A stream is composed of a header segment (expanded on the right) and a data segment.

would seem desirable to widen the data path used for configuration information.

There are attempts to widen the configuration bottleneck by incorporating a truly parallel data path for programming information. The Xilinx XC6200 series [4] and the National Semiconductor CLAy [5] series use a random access method for reconfiguration. The configuration cells for these devices can be accessed in a similar way as cells in a standard RAM. An on-chip row/column address is presented to the device, and the programming information is either read from or written to the desired cells. This solves many of the problems associated with serially configurable FPGAs. Partial reconfiguration is supported, and, to an extent, configuration time is lessened through the use of an 8-bit configuration path for the CLAy, and a 32-bit configuration path for the XC6200. While these improvements are welcomed, there are still shortcomings. One fundamental limitation of the design is the implied used of a centralized control scheme. Serially configurable devices suffer from this downfall as well: only one controller at a time can configure the device through the access port. While access to that port can be time multiplexed there is still only one data path used for configuring the device at any given time. Further, in terms of silicon area, the infrastructure needed to support the random access approach can be quite expensive due to the global scope of the routing and control logic required.

What is needed is a distributed control scheme in which multiple independent computational streams can configure the system simultaneously through multiple access ports. One advantage of a distributed scheme is scalability. The global controller in a RAM-based or serial design becomes a bottleneck for the system, being limited by the amount of time needed to access the various configurable resources in the system as well as the controller's ability to simultaneously schedule and program separate jobs effectively. As the size of the system grows, the demands on a global controller become greater and greater until it is the limiting factor in performance. Further, the communication latency inherent in a large system limits the speed with which configuration data can be distributed using a RAM based approach.

2. WORMHOLE RTR

Wormhole Run-Time Reconfiguration (RTR) attempts to address the weaknesses of FPGAs when used in a computational environment, and to provide a framework for implementing large-scale rapid run-time reconfigurable CCM platforms. It is intended as a method of rapidly creating and modifying custom computational pathways using a distributed control scheme (*data-driven* partial run-time reconfiguration). Similar to the data movement process in data-flow machines, the onset or completion of computational streams are used to instigate the process of reconfiguration. However, Wormhole RTR is not limited to data flow computation and, as discussed below, can be adapted to work with other computing paradigms. The essence of the Wormhole RTR concept is formed from independent self-steering *streams* of programming information and operand data that interact within the architecture to perform the computational problem at hand. The method of computation itself can be compared to that of Pipenets, as discussed by Hwang, et. al. [6], in which multiple intersecting pipelines (streams) are formed within the processor to perform a task. One application of Wormhole RTR would be as a high speed configuration methodology for such a system.

Wormhole RTR is based on the stream concept, which is an extension of the more common definition of the term. In this case, a stream is a concatenation of a programming header and operand data. The programming header is used to configure a computational pathway through the system as well as to configure the operations to be performed by the various computational elements along the path. The stream is self-steering and, as it propagates through the system, configuration information is stripped from the front of the header and is used to program the unit at the head of the stream; thus, the size of the header diminishes as the stream propagates through the system. The stream header is composed of an arbitrary number of packets of programming information. Each packet contains all the information needed to configure a designated unit in the system. The composition and lengths of the packets are variable so that different packet types may coexist within the same stream header and hence heterogeneous unit types may be traversed by a given stream. The term *wormhole* is used here, as it is used in the computer networks community, to describe the method in which data is passed from one computational node to another (wormhole routing) [7].

Targeted mainly at DSP-type operations, the *Colt* integrated circuit is a prototype Wormhole RTR device which adopts the stream processing paradigm. Colt is a coarse-grain "FPGA" tailored to suit the computational operations and pathways commonly associated with 1-D and 2-D signal processing. This particular implementation contains a mixture of 16-bit and 1-bit programmable datapaths.

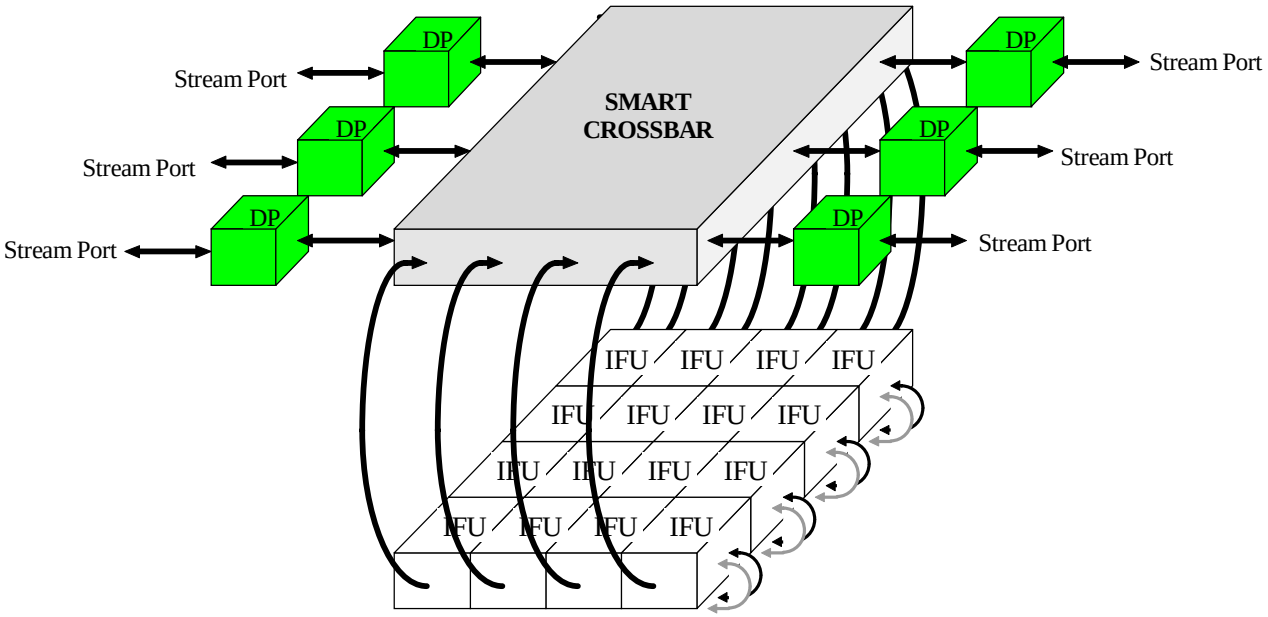

Figure 2 - The basic architecture of the Colt CCM, including Interconnected Functional Units (IFUs), Data Ports (DPs) and the Multiplier (MULT).

Figure 1 shows a stream that may be used with the Colt CCM (Figure 2), as fully described in another paper [8]. The Colt prototype IC consists of six 16-bit bi-directional data ports, a 4×4 toroidal mesh of Functional Units (FUs) and a 16-bit × 16-bit multiplier producing 32-bit results; all of which are connected through a "smart" crossbar network. Each of these units strips a packet from the front of an arriving stream header and stores configuration information from it. As shown, the stream header contains packets to program the various units of the computing platform in the order that they would be encountered along a path. In this case, the first packet is used to configure a data port, and then a crossbar packet that directs the stream to a particular column of the mesh. Once in the mesh, four separate packets configure Functional Units to perform independent functions on the data section of the stream. Finally, a second crossbar packet directs the stream out another data port, which is configured with the last packet in the stream header. Following the stream header is the data to be processed along that path through the Colt.

Note that the stream can be of arbitrary length. Thus, the path made through the system by the stream header can be arbitrarily long, allowing deep pipelines to be easily formed across multiple devices as shown in Figure 4. Likewise, the data section of the stream can also be arbitrarily long (such as could be found with the stream of values emanating from an A/D converter); allowing a single configuration to process multiple operand sets without reconfiguration. This ability again can be likened to the fetch/execute cycle of a microprocessor where the ratio of fetches to executions is generally 1:1. However, in a system using stream processing an 1:X ratio is achieved where X is the number of operand pairs that may be processed without reconfiguration. This represents a significant savings in overhead that would normally be associated with fetching opcodes. Further, a savings in power is realized because the state of the configuration information need not change between operand pairs.

The exact path taken by the stream through the system can be determined at run-time, allowing several competing processes to allocate resources as they become available. The process of resource allocation is not unlike that found in operating systems employing Banker's Algorithm to allocate I/O resources. A similar method could be used in this situation. However, in order to maintain maximum flexibility, this was not directly implemented on the Colt. Instead, this function will be implemented externally by Stream Controllers that arbitrate for resources both on and off-chip and substitute parameters for the physical resources allocated at run-time into the stream header. The process is similar to a normal operating system loading an executable and substituting for relocatable addresses. The design of such controllers is an area currently under scrutiny at Virginia Tech.

3. DISTRIBUTED CONTROL

Because the streams independently guide themselves through the system using the information contained in the stream header, the configuration process is inherently distributed. Multiple independent streams can wind their way through the Colt chip simultaneously. The streams can all originate from independent and distinct external controllers, or streams can be initiated by the onset or completion of other streams in the system through the Stream Controllers. Likewise, part of the system can be used to process data operands while any set of neighboring units can be accepting configuration data from the header of one or more other streams, thus allowing overlap of the configuration/execute cycle and of process multitasking within the same system.

These possibilities are all offered due to the distributed nature of Wormhole RTR.

The distributed control scheme offered by Wormhole RTR also provides advantages for system scalability. Communication operations during both phases (configuration and execution) of stream processing are localized in nature. Thus, long latencies between distant units within the system can be effectively hidden through natural pipelining, thus allowing the overall speed of the system to increase over that of a centrally controlled machine.

4. IMPLEMENTATION EXAMPLE

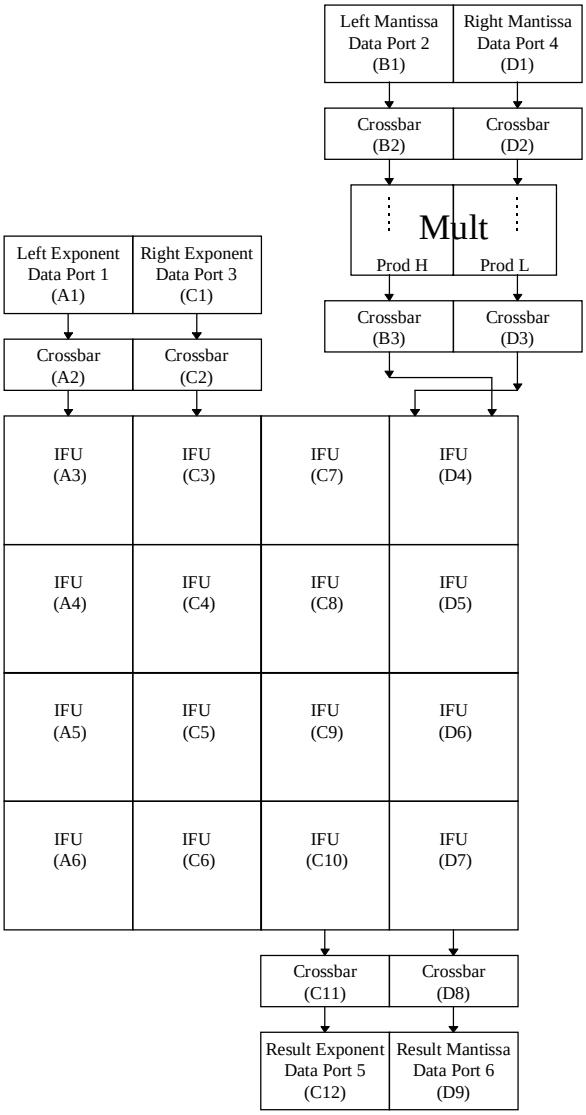

Figure 3 - Stream paths used to configure the floating point multiplier.

As an example of how Wormhole RTR can be used with the Colt CCM to perform useful computational functions, the implementation of a floating point multiplier is briefly discussed here. The format to be used for the floating point computations is represented by two 16-bit words, which can neatly be routed through the data paths of the Colt CCM. The first word contains a leading sign bit that is set to 0 for positive numbers and 1 for negative quantities. The remainder of the first word contains a 15-bit unbiased two's complement exponent. The second word contains a 16-bit mantissa. Note that this is only one possible representation and others may be used at the discretion of the programmer through the creation of other configurations.

Figure 3 shows an overview of the configuration paths used to program the floating point multiplier. Data ports 1 & 2 are used to inject the exponent and mantissa words, respectively, of the "left" floating point operand. Data ports 3 & 4 are used to inject the exponent and mantissa words of the "right" operand and the computed result is routed out of data ports 5 & 6. There are four streams shown: *A*, *B*, *C* and *D*; each of which consists of multiple packets of configuration information. Each of the units traversed by a stream is represented by a single box. The type of unit traversed is labeled, and the stream that configures that unit is indicated by the label in parenthesis; giving the letter of the stream that configures it and the number of the specific packet within the configuration header that will perform the configuration. This number gives the order that the packets appear in the configuration header, however, due to the pipelined nature of the configuration process, this may not necessarily indicate the specific order in which configuration takes place.

The multiplier implemented by this example is fully pipelined and can produce a new result on each clock cycle, giving an overall performance of 50 MFLOPs with the Colt CCM.

5. VERSATILITY OF UNIT COMPOSITION

The flexibility of the packet format within the stream header allows the Wormhole RTR concept to be applied to a much broader range of computing than simply CCMs. The information carried within a packet can vary from several dozen bits needed to program a computational element within an FPGA to an entire program to be executed on a standard microprocessor. Thus, the types of units along the path traversed by the stream through the system may be very diverse. The stream is a conduit for control information and data that can independently guide itself and perform a computational task in a heterogeneous computing environment. The localization of the operations to be performed to data items contained within the stream itself, or between several interacting streams, simplifies not only the burden of communications overhead, but also eases the task of integrating normally estranged technologies to achieve optimal computational performance. In this sense, Wormhole RTR could also be applied to a Macro Data Flow system as presented by Gaudiot, et. al. [9], in which a data flow graph topology is mapped onto a computational system consisting of a heterogeneous set of computational nodes of varying granularity.

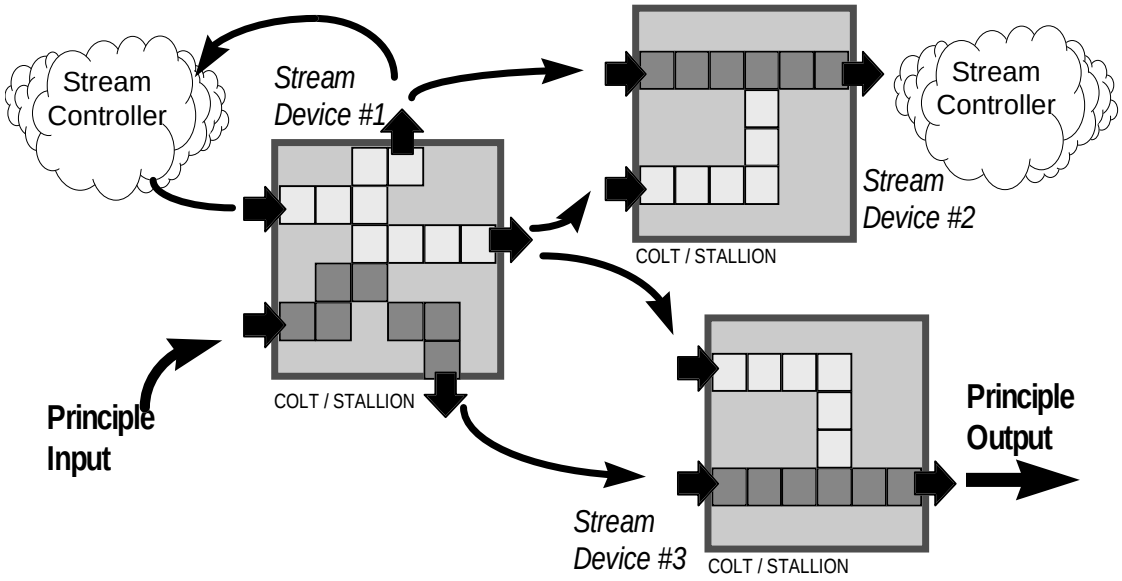

Figure 4 - The generalized wormhole RTR stream processing concept utilizing the Colt/Stallion integrated circuits. The small squares within the Colt/Stallion boxes represent resources allocated by the computational stream.

The Colt CCM itself is an amalgamation of different computational resources. Essentially, the basic elements of the Colt are standalone units that may be connected in a variety of ways by a stream. The crossbar is used to facilitate these connections and to maximize the flexibility given to the programmer as to the exact data flow graph implemented. There are no inherent interdependencies between the design of the different units of the Colt, however, and the combinations of use are limited only by the programmer's imagination. Linked with the data flow concept of expressing the graph such that only the computational dependencies of the algorithm are required, the Colt programmer may create an optimally parallel implementation of the algorithm. This is, of course, subject to the availability of resources, but in a full scale system resource restraints need not play a factor while, at the same time, all the inherent advantages of the Wormhole RTR methodology will still apply.

6. STREAM DESIGN PRINCIPLES

Indeed, the design of an architecture based around Wormhole RTR is conducive to concurrent processing since the units can be designed independently of one another. Unlike the slave-like execution of units in a microprocessor, the design of units for use in a Wormhole RTR architecture must be designed with a degree of intelligence in order to support the feed forward control strategy. This concept is best illustrated by the crossbar of the Colt. New stream(s) may enter any or all of the 12 inputs to the Colt crossbar at any time. Thus, the control mechanism of the crossbar must be a distributed system capable of simultaneously switching any set of inputs to a set of outputs, as well as supporting broadcast modes. To accomplish this, each of the crossbar outputs is assigned a unique address. Each switching point from input to output in the crossbar is designed with a small state machine. The state machine monitors the input to the crossbar for a stream header containing a packet designated for the output address that it is associated with. Upon receipt of such a packet, the state machine triggers the switching mechanism between the input and output and allows the remainder of the stream to propagate through. Because the state machines act independently of one another, the nightmare of coordinating all possible switching combinations is neatly avoided. The only point of common communication between the state machines occurs when an input has a pre-existing switch connection to a given output. Only one input may be forwarded to a given output at a time, thus when a new input requests the use of that output a clearing signal is sent to all state machines associated with that output, allowing the new input to gain exclusive control of the crossbar output. There are 156 such state machines in the Colt CCM. These are, however, quite small and reside compactly underneath the crossbar routing.

Thus, the concept of addressing was not totally forgone in the design of the Colt CCM. Theoretically, a Wormhole RTR system could be wholly designed around a relative addressing scheme. A relative addressing scheme in the mesh, for example, would use local cues for the direction of stream propagation from a given Functional Unit, such as north, south, east or west, or a combination of these. Obviously, the crossbar consists of 16 "cardinal" directions from which to proceed from input to output, but again, relative addressing could have been used. The advantage of a relative addressing scheme would be the same as that of relocatable code in a microprocessor. A stream could be injected into the Colt CCM and it could then map itself onto the hardware starting from any point. Care would need to be taken that independent streams did not collide by allocating the same resources in the process.

Though the Wormhole RTR concept can incorporate a relative addressing design strategy, it was decided that an address that was unique throughout the chip would be given to each unit. These addresses are used to distinguish units on a local basis, in contrast to a global strategy such as is used in a RAM. A unit will only respond to a packet containing either its unique address or the broadcast address. This was done to

better support and control the idea of broadcast programming and divergent streams. A divergent stream is created by configuring a unit to forward the stream in more than one direction simultaneously. The addresses of following packets in the stream header are carefully chosen to program selected units along the paths that have been newly created. This is one method through which a single stream can program multiple data paths. Though the stream is forwarded in multiple directions, the addresses allow the programmer to direct packets to specific units along the paths; thus, maintaining control over the individual configuration of each unit. Broadcast programming allows a single stream to diverge into two or more streams by programming units that are adjacent in the network with the same packet. This is useful for programming repetitious structures quickly. The addressing mechanism then allows the programmer to control the path that the stream will take to exit the structure once it has been configured.

7. OBJECT ORIENTED STREAMS

The localized nature of the communications involved with Wormhole RTR allow different sections of the data flow graph being implemented to expand or contract dynamically at run-time to utilize as much hardware as the current algorithm and existing system constraints allow. As different processes compete for hardware resources and the size of the physical machine increases or decreases, different portions of a given data flow graph could be implemented directly in hardware. For instance, if more hardware were added to a given system it could be detected by the Stream Controllers and used to configure more of the data flow graph directly in hardware, realizing an instant speedup with no modification or recompilation of the data flow "program" itself. As another example, the amount of physical resources in the system may remain the same, but a new process may be introduced. Existing processes on the machine could dynamically shrink the amount of code that is directly implemented in hardware to allow room for the new process to execute in a reasonable amount of time.

Figure 5 shows the extension of this concept of dynamic process expansion and shrinking to allow the possibility of object oriented streams in which the stream header simply contains "opcodes," each of which indicate the type of operation to be performed without necessarily dictating the exact piece of hardware to be used for implementation. A Stream Controller could receive such a stream, compare the opcode against a library of possible physical implementation strategies and then implement the most efficient method possible given available hardware resources by substituting the appropriate implementation into the stream header in place of the opcode.

In this way, a Wormhole RTR system could also implement the concept of generalized hardware in which the exact hardware to be utilized at run-time need not be known at compile time. This applies not only to hardware resource availability, such as would be used for the dynamic expansion and shrinkage discussed, but it also opens the possibility of running a program on future hardware that has yet to be added to the system, or has yet to even be invented, without having to recompile or recode. All that would be required is the addition of the implementation library for the new hardware to the system; dynamic stream to hardware mapping by the Stream Controllers could then immediately utilize it at run-time.

8. CONCLUSIONS

The immediate benefits of Wormhole RTR can be seen in the implementation of the Colt CCM. Running at 50 MHz, the six 16-bit data ports of the Colt can all be used to simultaneously transfer 4.8 billion bits of configuration information per second through the I/O pins. In comparison, the Xilinx XC4025 is capable of 10 million bits per second and the Xilinx XC6216 can transfer 800 million bits per second. This high speed is owed to three major improvements of the Colt architecture over previous CCM and FPGA type devices. The first of these is the truly word-wide configuration path made feasible by the Wormhole RTR concept of transmitting configuration data over the same signal paths used for operand data. While a word-wide configuration path could be used to

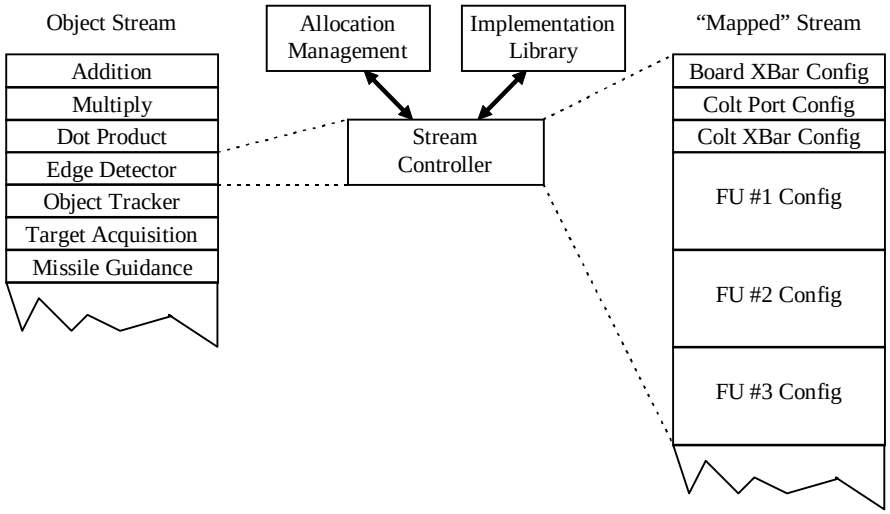

Figure 5 - Object oriented stream concept.

configure other devices, Wormhole RTR eases the burden by allowing the same signal path to be used, saving silicon area over the alternative of implementing twin bus structures, one for configuration and one for operand data. The second major reconfiguration speedup is derived from the high clock rate possible due to the localized nature of communications within the Colt CCM. A global control strategy, such as a random-access based approach, requires a large network of signals to allow any part of the chip to be accessed at any given time from the point of centralized control. By limiting the access to a localized region of the device, shorter propagation times can be achieved and higher clock rates realized. The third benefit of Wormhole RTR that contributes to the performance of the Colt CCM is distributed control which allows all six data ports to be used to configure the device simultaneously. By alleviating the bottleneck of a single injection point for configuration data, all the advantages of parallelism come into play. It should be stressed that the Colt is the first prototype of its kind and that the benefits reaped from all these factors will be magnified in planned devices.

The net result of going to a word-oriented approach over a bit-oriented approach is a gain in computational density over traditional bit-oriented FPGAs. The gain in density is partly spatial, owing to the fact that fewer switches, storage elements and control logic circuits are required. The gain is also temporal, in that fewer configuration bits are required allowing faster reconfiguration times. Finally, because fewer switching elements need to be traversed to perform a computation, the raw clock speed of the Colt can be higher, boosting the computing power even further. This is not to imply that word-oriented approaches are inherently superior to bit-oriented approaches; there are numerous instances and applications where bit-oriented solutions are obviously better.

Further information can be found in Bittner's dissertation covering Wormhole RTR, the Colt CCM and related material [10].

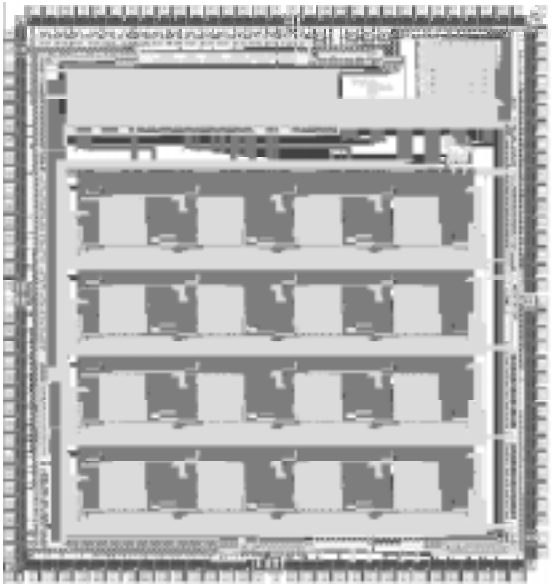

Figure 6 - A picture of the Colt configurable computing integrated circuit.

9. ACKNOWLEDGMENTS

The authors would like to thank the students who contributed to this project, including Mark Cherbaka, Brian Kahne, Mark Musgrove, Tsung-Han Yang, and Scott Harper. This work was funded by grant J-FBI-94-219 from the DARPA Information Technologies Office.

10. REFERENCES

[1] J. Eldredge and B. Hutchings, "Density Enhancement of a Neural Network Using FPGAs and Run-Time Reconfiguration," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, edited by D. Buell and K. Pocek, Napa, CA, pp. 180-188, April 1994.

[2] *The Programmable Logic Data Book*. Xilinx Incorporated, San Jose, California, pp. 2-4 - 2-43, 1994.

[3] *Altera 1995 Data Book*. Altera Corporation, San Jose, California, pp. 37-93, 1995.

[4] A. Stansfield and Ian Page, "The Design of a New FPGA Architecture," in *Proceedings of the Forth International Workshop on Field Programmable Logic*, Oxford, England, Springer-Verlag, New York, New York, September, 1995.

[5] C. Rupp, "CLAyFun Reference Manual," National Semiconductor Corporation, Santa Clara, California, July 1995.

[6] K. Hwang, *Advanced Computer Architecture*, pp. 442-446, McGraw-Hill, New York, New York, 1993.

[7] W. Dally, C. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks," IEEE Transactions on Computing, C-36, no. 5, pp. 547-553, May 1987,

[8] R. Bittner, M. Musgrove, P. Athanas, "Colt: An Experiment in Wormhole Run-Time Reconfiguration," to appear at Photonics East, Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using FPGAs, Boston, MA, November, 1996.

[9] J. L. Gaudiot and M. D. Ercegovac. Performance Analysis of a Data-Flow Computer with Variable Resolution Actors, *Proceedings of the 4th International Conference on Distributed Computing Systems*, The Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ, pp. 2-9, 1984.

[10] R. Bittner, "Development and VLSI Implementation of a High Speed Data Flow DSP Computing System," Ph.D. Dissertation, Bradley Department of Electrical and Computer Engineering, Virginia Tech, 1996, work in progress.

[11] P. Athanas, I. Howitt, T. Rappaport, J. Reed, and B. Woerner, "A High Capacity Adaptive Wireless Receiver Implemented with a Reconfigurable Computer Architecture", ARPA GloMo Principle Investigators Conference, San Diego, CA, November 1995.

[12] M. Bove, J. Watlington, “Cheops: A Reconfigurable Data-Flow System for Video Processing,” *IEEE Trans. on Circuits and Systems for Video Processing*, no. 5, pp. 140-149, April 1995.