My Profile Log In

HOME
ABOUT US
CONTACT US
HELP

Home / Engineering / Electrical and Electronics Engineering

Wiley Encyclopedia of Electrical and Electronics Engineering

Reconfigurable Computing

Standard Article

Cheng Chung-Kuan[1], Andrew B. Kahng[2], Philip H.W. Leong[2]

[1]Dept. of Computer Science and Engineering, University of California, La Jolla, California

[2]Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong

Copyright © 2007 by John Wiley & Sons, Inc. All rights reserved.
DOI: 10.1002/047134608X.W7603.pub2
Article Online Posting Date: June 15, 2007

Recommend to Your Librarian
Save title to My Profile
Email this page
Print this page

BROWSE THIS TITLE
Article Titles A–Z
Topics

SEARCH THIS TITLE
Go
Advanced Product Search
Search All Content
Acronym Finder

Abstract | Full Text: HTML PDF (1550K)

Abstract

Reconfigurable computing is the application of adaptable fabrics to solve computational problems, often taking advantage the flexibility available in the fabric to produce problem-specific architectures that achieve high performance because of customization. Reconfigurable computing has been successfully applied to fields as diverse as digital signal processing, cryptography, bioinformatics, logic emulation, CAD tool acceleration, scientific computing, and rapid prototyping.

Although Estrin-first proposed the idea of a reconfigurable system in the form of a fixed plus variable structure computer in 1960 (1) it has only been in recent years that reconfigurable fabrics have reached sufficient density to make them a compelling implementation platform for high Performance applications and embedded systems. In this article, intended for the non-specialist, we describe some of the basic concepts, tools and architectures associated with reconfigurable computing.

Keywords: reconfigurable computing; adaptable fabrics; application integrated circuits; field programmable gate arrays (fpgas); system architecture; runtime

About Wiley InterScience | About Wiley | Privacy | Terms & Conditions
Copyright © 1999-2007John Wiley & Sons, Inc. All Rights Reserved.

RECONFIGURABLE COMPUTING

INTRODUCTION

Although reconfigurable fabrics can in principle be constructed from any type of technology, in practice, most contemporary designs are made using commercial field programmable gate arrays (FPGAs). An FPGA is an integrated circuit containing an array of logic gates in which the connections can be configured by downloading a bitstream to its memory. FPGAs can also be embedded in integrated circuits as intellectual property cores. More detailed surveys on reconfigurable computing are available in the literature (2–5).

Microprocessors offer an easy-to-use, powerful, and flexible implementation medium for digital systems. Their utility in computing applications makes them an overwhelming first choice, and parallel interconnections of microprocessors can be extremely powerful. Moreover, it is relatively easy to find software developers, and microprocessors are widely supported by operating systems, software engineering tools, and libraries. Unfortunately, their generality does not make them the best choice for a large class of applications that need to be optimized for performance, power, board area.

Application-specific integrated circuits (ASICs) and FPGAs are able to arrange computations in a spatial rather than temporal fashion and greater levels of parallelism than a microprocessor can be achieved. Thus, performance improvements of several orders of magnitude can be achieved. Also, the absence of caches and instruction decoding can result in the same amount of work being done with less chip area and lower power consumption (6). As an example, in a cryptographic key search problem, a single FPGA with 96 parallel RC4 encryption engines operating at 50 MHz achieved a speedup of 58 over a 1.5-GHz Pentium 4 implementation (7).

An example involving the implementation of a finite impulse response (FIR) filter is shown in Fig. 1. The reconfigurable computing solution is significantly more parallel than the microprocessor-based one. In addition, it should be apparent that the reconfigurable solution avoids the overheads associated with instruction decoding, caching, register files, and speculative execution, and unnecessary data transfers as well as control hardware can be omitted.

Compared with ASICs, FPGAs offer very low nonrecurrent engineering (NRE) costs, which is often a more important factor than the fact that FPGAs have higher units costs and many applications do not have the very high volumes required to make ASICs a cheaper proposition. As integrated circuit feature sizes continue to decrease, the NRE costs associated with ASICs continue to escalate, increasing the volume at which it becomes cheaper to use an ASIC (see Fig. 2). Reconfigurable computing will be used in increasingly more applications, as ASICs become only cost effective for the highest performance or highest volume applications.

Additional benefits of reconfigurable computing are that its technology provides a shorter time to market than ASICs (associated FPGA fabrication time is essentially zero), making many fabrication iterations within a single day possible. This benefit allows more complex algorithms to be deployed and makes possible problem-specific customizations of designs. FPGA-based designs are inherently less risky in terms of technical feasibility and cost, as shorter design times and lower upfront costs are involved. As its name suggests, FPGAs also offer the possibility of modifications to the design in the field, which can be used to provide bug fixes, modifications to adapt to changing standards, or to add functionality, all of which can be achieved by downloading a new bitstream to an existing reconfigurable computing platform. Reconfiguration can even take place while the system is running, this being known as runtime reconfiguration [e.g., (8)]. Runtime reconfiguration is explained in more detail later in this article.

In the next section, we introduce the basic architecture of common reconfigurable fabrics, followed by a discussion of applications of reconfigurable computing and system architectures. Runtime reconfiguration and design methods are then covered. Finally, we discuss multichip systems and end with a conclusion.

RECONFIGURABLE FABRICS

A block diagram illustrating a generic fine-grained island-style FPGA is given in Fig. 3 (9). Products from companies such as Xilinx (10) Altera (11), and Actel (12) are commercial examples. The FPGA consists of a number of logic cells that can be interconnected to other logic and input/output (I/O) cells via programmable routing resources. Logic cells and routing resources are configured via bit-level programming data, which is stored in memory cells in the FPGA. A logic cell consists of user-programmable combinatorial elements, with an optional register at the output. They are often implemented as lookup tables (LUTs) with a small number of inputs, 4-input LUTs being shown in Fig. 3. Using such an architecture, subject to FPGA-imposed limitations on the circuit's speed and density, an arbitrary circuit can be implemented. The complete design is described via the configuration bitstream which specifies the logic and I/O cell functionality, and their interconnection.

Current trends are to incorporate additional embedded blocks so that designers can integrate entire systems on a single FPGA device. Apart from density, cost, and board area benefits, this process also improves performance because more specialized logic and routing can be used and all components are on the same chip. A contemporary FPGA commonly has features such as carry chains to enable fast addition; wide decoders; tristate buffers; blocks of on-chip memory and multipliers; embedded microprocessors; programmable I/O standards in the input/output cells; delay locked loops; phase locked loops for clock deskewing, phase shifting and multiplication; multi-gigabit transceivers (MGTs); and embedded microprocessors. Embedded microprocessors can be implemented either as soft cores using the internal FPGA resources or as hardwired cores.

In addition to the architectural features described, intellectual property (IP) cores, implemented using the logic cell

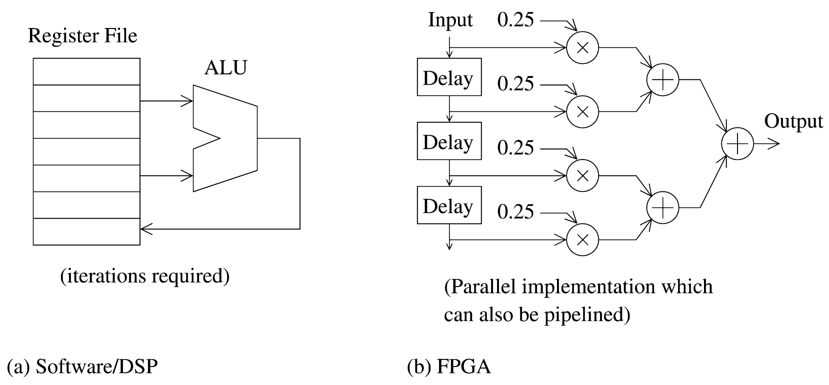


Figure 1. Illustration of a microprocessor based FIR filter vs. a reconfigurable computing solution. In the microprocessor, operations are performed in the ALU sequentially. Furthermore, instruction decoding, caching, speculative execution, control generation and so on are required. For the reconfigurable computing approach using an FPGA, spatial composition is used to increase the degree of parallelism. The FPGA implementation can be further parallelized through pipelining.

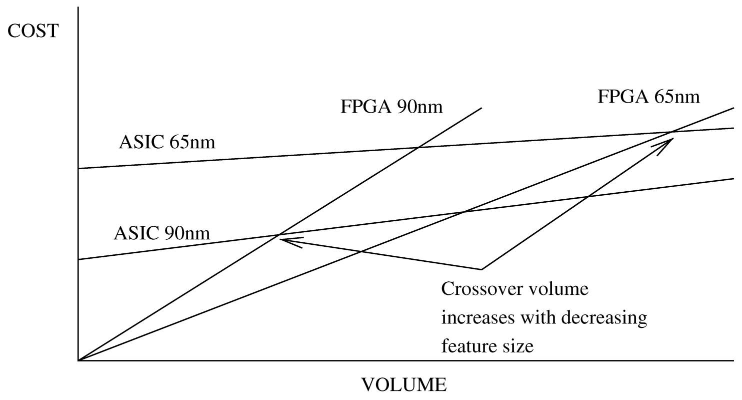


Figure 2. Cost of technology vs. volume. The crossover volume for which ASIC technology is cheaper than FPGAs increases as feature size is reduced because of increased non-recurrent engineering costs.

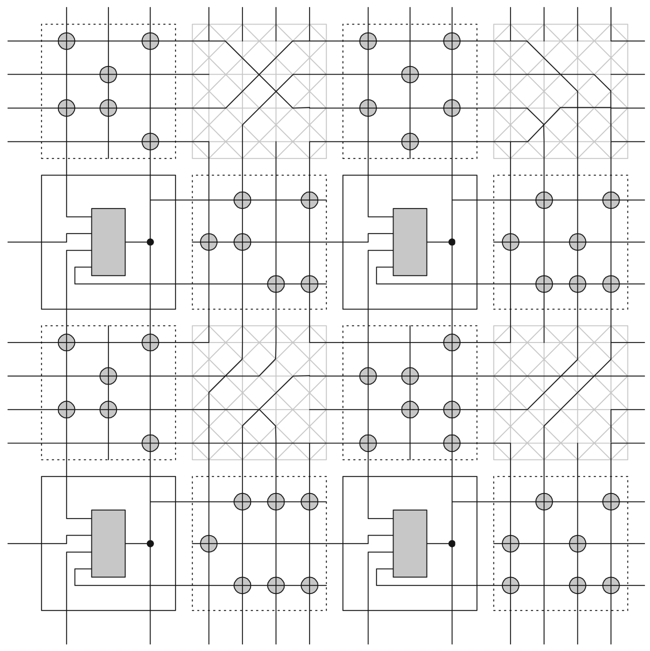


Figure 3. Architecture of a basic island-style FPGA with four-input logic cells. The logic cells, shown as gray rectangles are connected to programmable routing resources (shown as wires, dots, and diagonal switch boxes) (source: Reference (9) and (76)).

resources of the FPGA, are available from vendors and can be incorporated into a design. These IP cores include bus interfaces, networking components, memory interfaces, signal processing functions, microprocessors and so on and can significantly reduce development time and effort.

The bit-level organization of the logic and routing resources in island-style FPGAs is extremely flexible but has high implementation overhead as a result. Tradeoffs exist in the granularity of the logic cells and routing resources. Fine-grained devices have the best flexibility; however, coarse-grained elements can trade some flexibility for higher performance and density (13). With modern technologies, the speed of the routing resource is a limiting factor and trends have been to increase the functionality of the logic cells (e.g., use logic cells with larger numbers of inputs to reduce interconnect requirements). For datapath oriented applications such as in digital signal processing, coarse-grained architectures such as Pipewrench (14) and RaPID (15) employ bus-based routing and word-based functional units to utilize silicon resources more efficiently.

Companies such as M2000 (16) and eASIC (l7) provide reconfigurable fabric in the form of an IP block that can be embedded in an ASIC design. Fully synthesizable embedded cores have also been proposed (18). Such fabrics enable post-fabrication changes and allow the flexibility and performance benefits associated with reconfigurable computing to be enjoyed in ASICs. They will be increasingly important in future systems for adding functionality, fixing bugs, monitoring, and debugging after fabrication.

APPLICATIONS

Reconfigurable computing has found widespread application in the form of "custom computing-machines" for high-energy physics (19), genome analysis (20), signal processing (21, 22), cryptography (7, 23), financial engineering (24) and other domains (25). It is unique in that the flexibility of the fabric allows customization to a degree not feasible in an ASIC. For example, in an FPGA-based implementation of RSA cryptography (23), a different hardware modular multiplier for each prime modulus was employed (i.e., the modulus was hardwired in the logic equations of the design). Such an approach would not be practical in an ASIC as the design effort and cost is too high to develop a different chip for different moduli. This led to greatly reduced hardware and improved performance, the implementation being an order of magnitude faster than any reported implementation in any technology at the time.

Another important application is logic emulation (26, 27) where reconfigurable computing is used not only for simulation acceleration, but also for prototyping of ASICs and in-circuit emulation. In-circuit emulation allows the possibility of testing prototypes at full or near-full speed, allowing more thorough testing of time-dependent applications such as networks. It also removes many of the dependencies between ASIC and firmware development, allowing them to proceed in parallel and hence shortening development time. As an example, it was used in Reference (28) for the development of a two-million-gate ASIC containing an IEEE 802.11 medium access controller and IEEE 802.1 la/b/g physical layer processor. Using a reconfigurable prototype of the ASIC on a commodity FPGA board, the ASIC went through one complete pass of real-time beta testing before tape-out.

Digital logic, of course, maps extremely well to fine-grained FPGA devices. The main design issues for such systems lie in partitioning of a design among multiple FPGAs and dealing with the interconnect bottleneck between chips. The Cadence Palladium II emulator (29) is a commercial example of a logic emulation system and has 256-million-gate logic capacity and 74-GB memory capacity. It uses custom ASICs optimized for logic emulation and is 100–10,000 times faster than software-based register transfer language simulation. Further discussion of interconnect time-multiplexing and system decomposition is given later in this article.

Hoang (20) implemented algorithms to find minimum edit distances for protein and DNA sequences on the Splash 2 architecture. Splash 2 can be modeled in terms of both bidirectional and unidirectional systolic arrays. In the bidirectional algorithm, the source character stream is fed to the leftmost processing element (PE), whereas the target stream is fed to the rightmost PE. Comparing two sequences of length m and n requires at least $2 \times \max(m + 1, n + 1)$ processors, and the number of steps required to compute the edit distance is proportional to the size of the array. The unidirectional algorithm is suited for comparing a single source sequence against multiple target sequences. The source sequence is first loaded as in the bidirectional case, and the target sequences are fed in one after the other and processed as they pass through the PEs (which results in virtually 100% utilization of processors, so that the unidirectional model is better suited for large database searches).

The BEE2 system (22), described in the next section, was applied to the radio astronomy signal processing domain, which included development of a billion-channel spectrometer, a 1024-channel polyphase filter banks, and a two-input, 1024-channel correlator. The FPGA-based system used a 130-nm technology FPGA and performance was compared with 130-and 90-nm DSP chips as well as a 90-nm microprocessor. Performance in terms of computational throughput per chip was found to be a factor of 10 to 34 over the DSP chip in 130-nm technology and 4 to 13 times better than the microprocessor. In terms of power efficiency, the FPGA was one order of magnitude better than the DSP and two orders of magnitude better than the microprocessor. Compute throughput per unit chip cost was 20–307% better than the 90-nm DSP and 50–500% better than the microprocessor.

SYSTEM ARCHITECTURES

Reconfigurable computing machines are constructed by interconnecting one or more FPGAs. Functionally, we can view FPGA-based systems as consisting of two components, reprogrammable FPGAs providing logic implementation and field programmable interconnect chips (FPICs) providing connectivity among FPGAs. The FPICs, in turn, could be implemented as ASICs or using FPGAs. Most systems include other elements, such as microprocessors and storage, and can be treated as processing elements and memory that are interconnected. Obviously, the arrangement of these elements affects the system performance and routability.

The simplest topology involves FPGAs directly connected in a ring, mesh, or other fixed pattern. FPGAs serve as both logic and interconnect, providing direct communication between adjacent devices. Such an architecture is predicated on locality in the circuit design and further assumes that the circuit design maps well to the planar mesh. This architecture fits well for applications with regular local communications (30). However, in general, high performance is hard to obtain for arbitrary communication patterns because the architecture only provides direct communications between neighboring FPGAs and two distant FPGAs may need many other devices as “hops” to communicate, resulting in long and widely variable delays. Furthermore, FPGAs, when used as interconnects, often result in poor timing characteristics.

A major change in the architecture of FPGA-based systems was the concept of a partial crossbar interconnect, as in Realizer (26) and BORG (31). This scheme is common in logic emulation systems. Interconnection through FPICs implies that all pairs of FPGAs are neighbors, resulting in predictable interconnect delays, better timing characteristics, and better overall system performance (32, 33). Figure 4, from Reference (26) depicts a reconfigurable computing system designed for logic emulation. Arrays of reconfigurable processors and FPICs, both implemented using FPGAs, reside on the emulation modules. The user inputs the emulated design netlist and commands from the workstation. The workstation and control processor personalize the emulation module, which are used in place of the emulated chip. Thus, the target system can function properly before the actual chip is available. Furthermore, testing and design change can be made by modifying software instead of reworking hardware.

Figure 5 depicts the SPLASH 2 architecture (34). Each board contains 16 FPGAs, X1 through X16. The blocks M1 through M16 are local memories of the FPGAs. A simplified 36-bit bus crossbar, with no permutation of the bit-lines within each bus, interconnects the 16 FPGAs. Another 36-bit bus connects the FPGAs in a linear systolic fashion. The local memories are dual ported with one port connecting to the FPGAs and the other port connecting to the external bus. It is interesting to note that the crossbar was added to the SPLASH 2 machine, the original SPLASH 1 machine only having the linear connections. SPLASH 2 has been successfully used for custom computing applications such as search in genetic databases and string matching (20).

Other designs have used a hierarchy of interconnect schemes, differing in performance. The use of multi-gigabit transceivers (MGT) available on contemporary FPGAs allows high bandwidth interconnection using commodity components. An example is the Berkeley Emulation Engine 2 (BEE2) (22), designed for reconfigurable computing and illustrated in Fig. 6. Each compute module consists of five FPGAs (Xilinx XC2VP70) connected to four double data rate 2 (DDR2) dual inline memory modules (DIMMs) with a maximum capacity of 4GB per FPGA. Four FPGAs are used for computation and one for control. Each PPGA has two PowerPC 405 processor cores. A local mesh connects the computation FPGAs in a 2-D grid using low-voltage CMOS (LVCMOS) parallel signaling. Off-module communications are of via 18 (two from the control FPGA and four from each of the compute FPGAs) Infiniband 4*X* channel-bonded 2.5-Gbps connectors that operate full-duplex, which corresponds to a 180-Gbps off-module full-duplex communication bandwidth. Modules can be interconnected in different topologies including tree, 3-D mesh, or crossbar. The use of standard interfaces allows standard network switches such as Infiniband and 10-Gigabit Ethernet to be used. Finally, a 100 base-T Ethernet connection to the control FPGA is present for out-of-band communications, monitoring, and control.

Commercial machines such as the Cray XD1 (35), SRC SRC-7 (36), and Silicon Graphics RASC blade (37), have a similar interconnect structure to the BEE2 in that they are parallel machines employing high performance microprocessors tightly coupled to a relatively small number of FPGA devices per node. Nodes are interconnected via high speed switches and for specialized applications, such machines can have orders of magnitude performance improvement over conventional architectures. Switching topologies can be altered via configuration of the switching fabric.

RUNTIME RECONFIGURATION

A reconfigurable computing system can have its functionality updated during execution, resulting in reduced resource requirements. A runtime reconfigurable system partitions a design temporally so that the entire design does not need to be resident in the FPGA at any given moment (38, 39). Configuration and execution can be overlapped to improve performance in the presence of reconfiguration latency. Using this technique, designs that are larger than the physical hardware resources can be realized in an efficient manner.

Dharma, a time-sharing FPGA architecture, was proposed that contains a functional block and an interconnect network (40). The interconnect and the logic can be time-shared. The authors proposed that emulated design topology be levelized in a folded pipeline manner; this topology simplifies the architecture and provides predictable interconnect delay (Fig. 7).

Single context, partially reconfigurable, and multiple context architectures have been proposed. In a single context system, any changes to the functionality of the FPGA involves reloading the entire bitstream; early FPGAs were of this type. This scheme has the disadvantage of long reconfiguration time. Partial reconfiguration, as supported by the Xilinx Virtex FPGAs (10), allows portions of the FPGA to be changed via a memory mapped scheme, whereas the other portions of the FPGA continue functioning. Compared with a single context scheme, area overhead is associated in providing this feature. Multiple context architectures, such as NEC’s Dynamically Reconfigurable Processor (DRP) (41), allow a number of complete configurations to be stored in the fabric simultaneously and thus reconfiguration can be achieved in a small number of cy-

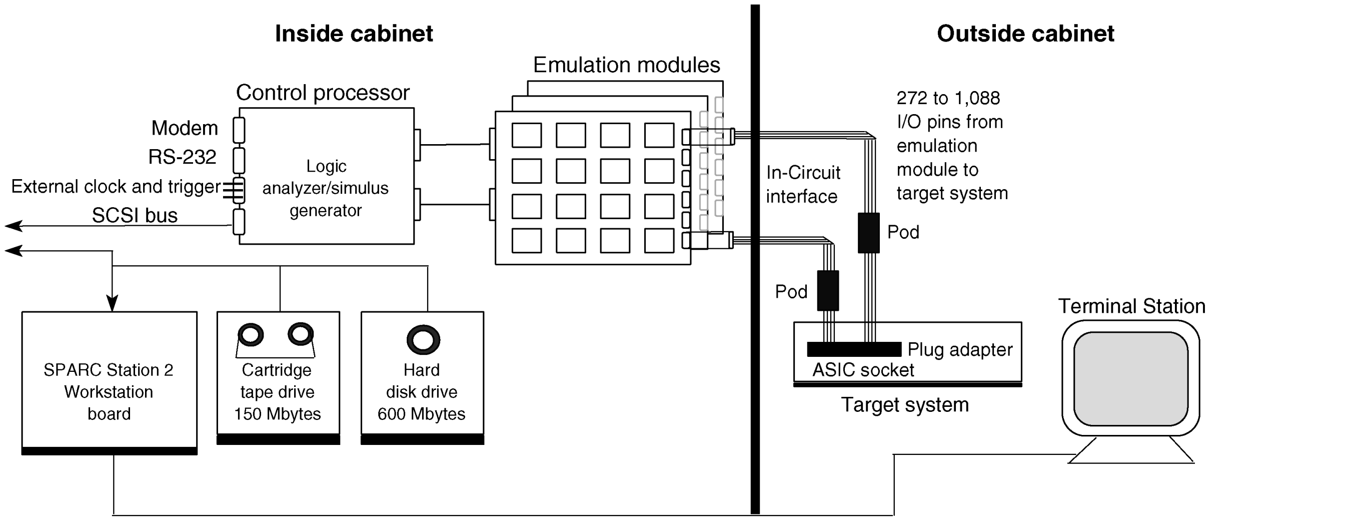

Inside cabinet
Outside cabinet
Emulation modules
Control processor
Modem
RS-232
External clock and trigger
SCSI bus
Logic analyzer/simulus generator
272 to 1,088 I/O pins from emulation module to target system
In-Circuit interface
Pod
Pod
Terminal Station
Plug adapter
ASIC socket
Target system
SPARC Station 2 Workstation board
Cartridge tape drive 150 Mbytes
Hard disk drive 600 Mbytes

Figure 4. Example of a logic emulation system. Arrays ofFPGAs and FPICs reside on the emulation modules. The user inputs the emulated design netlist and commands from the workstation. The workstation and control processor personalize the emulation modules, which are used in place of the emulated chip.

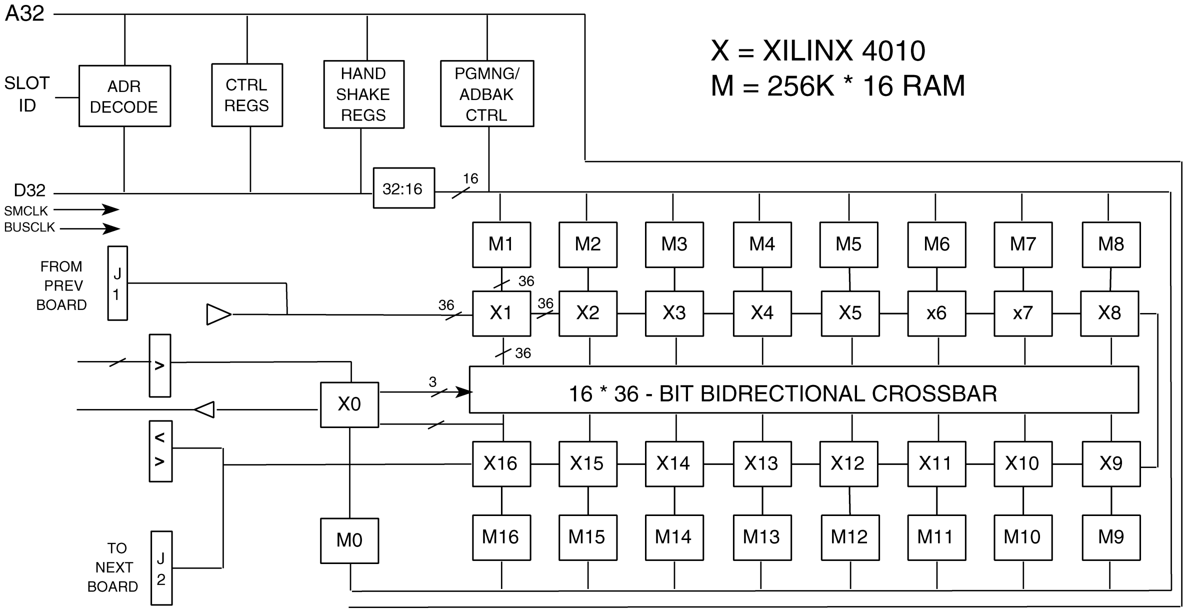


Figure 5. SPLASH2 architecture. Each board contains 16 FPGAs, XI through XI6. The blocks Ml through Ml6 are local memories of the FPGAs. A simplified 36-bit bus crossbar, with no permutation of the bit-lines within each bus, interconnects the 16 FPGAs. Another 36-bit bus connects the FPGAs in daisy-chain fashion. The local memories are dual ported with one port connecting to the FPGAs and the other port connecting to the external bus.

cles. This architecture has the shortest context switch time, however, a larger area overhead is associated with implementation of this scheme.

The logical unit of reconfiguration could be at a number of levels including the application, instruction, task, block, or sub-block level. An example of application-level reconfiguration could simply involve loading a runtime-dependent bitstream to support a particular coding standard in a video coding application. The Dynamic Instruction Set Computer (DISC) (42) supported demand-driven modification of the instruction set through partial reconfiguration. The commercial Stretch processor (43) combines reconfigurable fabric with a processor to support the execution of custom instructions implemented on a reconfigurable fabric. Furthermore, the fabric can be reconfigured at runtime and the design environment is software-centric, with programming of the processor being in Stretch C.

An operating system for guarantee-based scheduling of hard real-time tasks has been proposed (44). Under control of software running on a microprocessor, task circuits can be scheduled online and placed in a suitable free space in a hardware task area. Communications between tasks and I/O are done though a task communication bus, and termination of a task frees the reconfigurable resources used. It

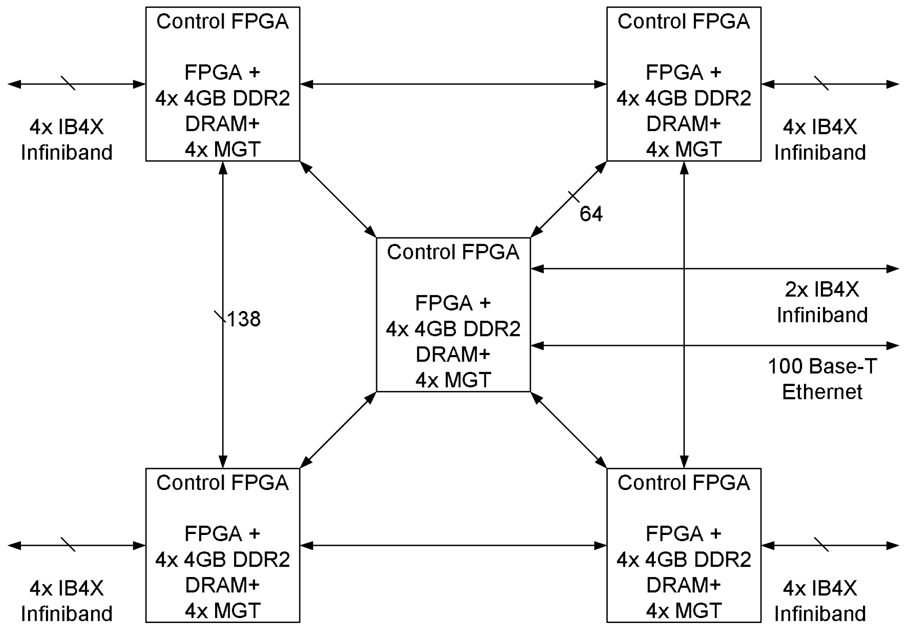


Figure 6. BEE2 Compute Module block diagram. Compute modules can be interconnected via the Infiniband IB4X connectors, either directly or via a 10-Gigabit Ethernet switch. The 100-Base T Ethernet can be used for control, monitoring, or data archiving.

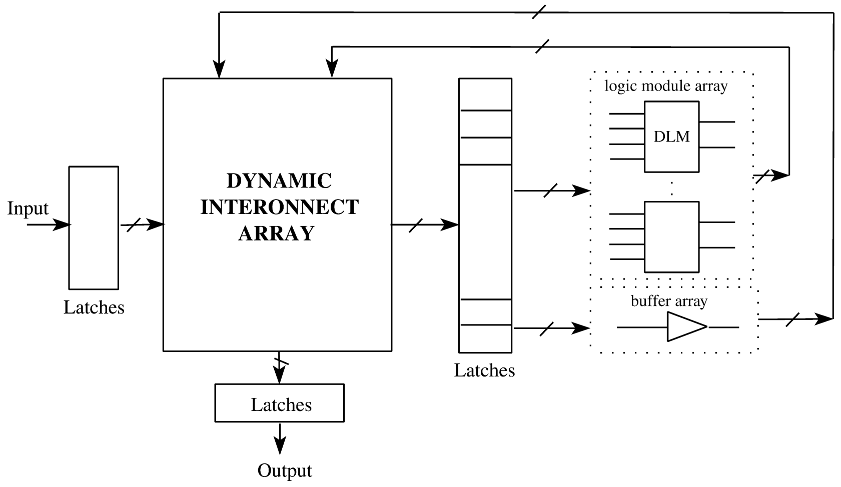


Figure 7. Dynamic Architecture for FPGA-based systems. The architecture contains a functional block and an interconnect network. The interconnect and the logic can be time shared. The emulated design topology is levelized in a folded pipeline manner. The levelized topology simplifies the architecture with predictable interconnect delay.

was shown that hardware in the hardware task area can be shared by tasks and the overheads associated with its implementation on a partially configurable platform were acceptably low.

A pipeline stage is often a convenient block-level unit for reconfiguration. In incremental pipeline reconfiguration (45), an application with S pipeline stages can be implemented in an FPGA with fewer than S physical pipeline stages. This is done by adding one pipeline stage and removing one pipeline stage in each stage of the computation. Execution and computation can be overlapped.

Runtime reconfiguration can be done at even lower levels. A crossbar switch which employs runtime reconfiguration of the FPGA's routing resources has been described (46). This scheme was able to achieve density, switch update latency and performance higher than possible using conventional means.

Tools have been developed to support runtime reconfiguration. For example, JBits (47) is a set of Java classes that provide an application programming interface to the Xilinx FPGA bitstream. The interface operates on either bitstreams generated by Xilinx design tools or on bitstreams read back from actual hardware and allows the FPGA logic and routing resources to be modified.

DESIGN METHODS

Hardware description languages (HDLs) such as the Very High Speed Integrated Circuit Hardware Description Language (VHDL) and Verilog are commonly used to specify the logic of a reconfigurable system. Descriptions in these languages have the advantage of being vendor neutral, so the same description can be synthesized for different targets such as different FPGA devices, different FPGA vendors, and ASICs. For this reason, these languages are often the target language for higher level tools that offer higher levels of abstraction.

Module generators and libraries are commonly deployed to promote reuse. For example, vendors such as Altera and Xilinx have parameterized libraries of components that can be used in a design. These libraries are generated so that a circuit optimized for the particular application can be produced. As an example, a parameterized floating point library might allow the wordlength of the exponent and mantissa to be specified as well as whether denormalized numbers are supported. The module generator then generates a netlist or VHDL-based floating point adder that can be included in a design.

A high level language can be directly mapped to a netlist or HDL. As an example, Luk and Page described a simple compilation process (8, 49) from a high level language with explicit parallel extensions to a register transfer language (RTL) description. Parallel execution of statements is implemented via parallel processes, and these can communicate via channels through which a single-word message can be passed. Variables in the user program are mapped to registers, all expressions are implemented as combinational logic, and multiplexers are used in the case a register has multiple sources. A datapath that matches the dataflow graph of the input source description is generated using this strategy. The clocking scheme employed is a global, synchronous one, and a convention that each assignment takes exactly one clock cycle is followed. A start signal is used to feed the clock and to enable each register that corresponds to a variable, and a finish signal is generated for the assignment in the following clock cycle. To execute statements sequentially, the start and finish signals of adjacent statements are simply connected together, creating a one-hot distributed control scheme. Conditional statements and loops are formed by asserting one of several possible start signals that correspond to alternative basic blocks in a program. Completion of conditional or loop constructs and synchronization of parallel blocks are implemented by combining relevant finish signals using the appropriate combinatorial logic. An example showing the translation of a simple code fragment to control and datapath is shown in Fig. 8.

Commercial tools that can compile standard programming languages such as Java, C, or C++ [e.g., (49)] are available. Examples include Handel-C from Celoxica (50) and Catapult C from Mentor Graphics (51). The use of traditional programming languages improves productivity as low level details are handled by the compiler. This is analogous to C versus assembly language for software development. Another difference with potentially large implication is that, using these tools, software developers can also design reconfigurable computing applications. Domain-specific languages such as MATLAB/Simulink (52) offer even greater improvements in productivity because they are interactive, include a large library of primitive routines and toolkits, and have good graphing capabilities. Indeed, many designs for communications and signal processing are first prototyped in MATLAB and then converted to other languages for implementation. Tools such as the MATCH compiler (53) and Xilinx System Generator can translate a subset of MATLAB/Simulink directly to an FPGA design.

The availability of embedded operating systems such as Linux for microprocessors on an FPGA provides a familiar software development environment for programmers, greatly facilitating program development through the availability of a large range of open-source libraries as well as high quality development tools. Such tools can greatly speed up the development time and improve the quality of embedded systems. Hardware/software codesign tools such as Altera's Nios II C-to-Hardware acceleration compiler enable time-critical functions in a C program to be converted to a hardware accelerator that is tightly coupled to a microprocessor within the FPGA (54).

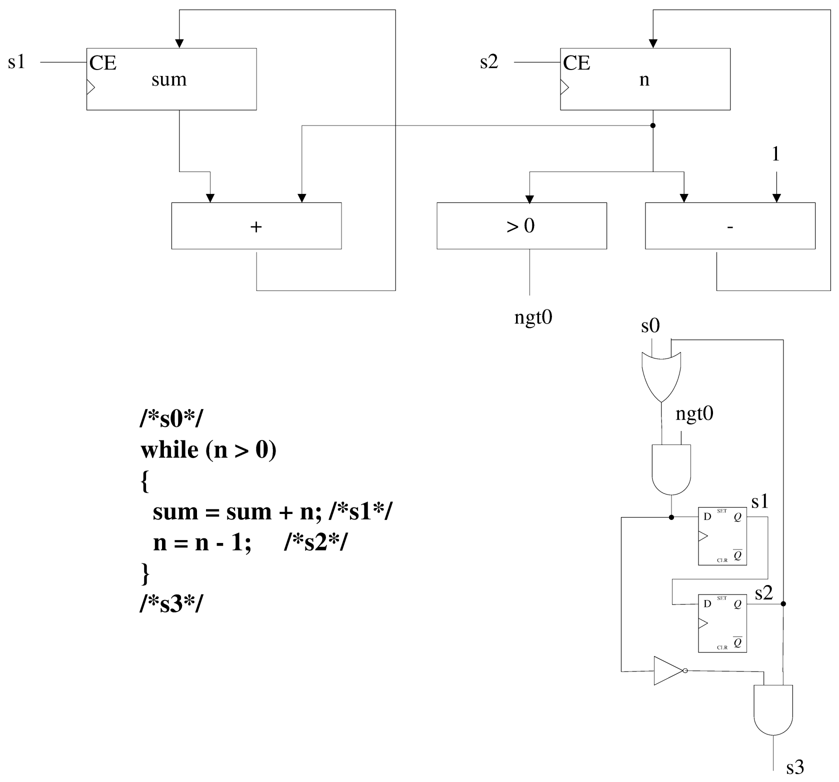
Issues developing with the mapping of algorithms to hardware are more generally discussed by Isshiki and Dai (55), who focus on the differences between implementing bit-serial versus bit-parallel modules (e.g., adders and multipliers) on FPGA architectures. Although latency is larger for bit-serial modules, the reduction in area frequently makes area-time products significantly lower for such implementations. More specifically, such advantages as the following can be obtained: 1) For bit-parallel modules, the I/O pin limitation is a major problem, and the large size of the module cluster can result in unused space and underutilized logic resources; 2) bit-serial modules are easier to partition as cell-to-cell connections are sparse and do not cause I/O problems; and 3) high fanout nets can impair routability of bit-parallel modules. Leong and Leong (56) generalized further with a design methodology that can translate a dataflow description with signals of different wordlengths to a digit serial design.

MULTICHIP SYSTEMS

Special care must be taken in the design of large and multichip reconfigurable systems. In this section, we describe some theoretic results relevant to the major architectural and issues associated with such designs.

Interconnect Organization

A classic Clos network (57) contains three stages: inputs, intermediate switches, and outputs, as shown in Fig. 9. It can be used to interconnect pins in a reconfigurable computing system, and its input and output stages are symmetric. Suppose the first stage has r $n\times m$ crossbar switches, the second stage has m $r \times r$ switches, and the third stage has r $m \times n$ switches, let us denote the network as $c(n,m,r)$. For any two-pin net interconnect requirement, the network $c(n, m, r)$ can achieve complete routability if m is not less than n. The routing method can be described by recursive operations (58). In the first iteration, we reduce the network to $c(n-1, m-1, r)$. In the *ith* iteration, we reduce the network to $c(n-i, m-i, r)$. When $n-i=1$, we have r $1 \times (m-n+1)$ switches in the first stage, $m-n+1$ $r \times r$



```
/*s0*/
while (n > 0)
{
  sum = sum + n; /*s1*/
  n = n - 1;     /*s2*/
}
/*s3*/
```

Figure 8. Hardware compilation example. The C program is translated into a datapath (top) and control (bottom). Execution of statements in the while loop are controlled by s1 and s2; s0 and s3 correspond to the start signals of the statements before and after the while loop.

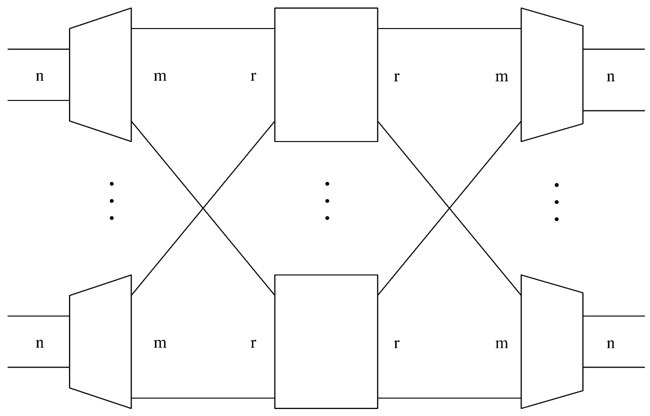


Figure 9. Clos network. A Clos network contains three stages: inputs, intermediate switches, and outputs. The input and output stages are symmetric. In the figure, the first-stage has r $n \times m$ switches, the second-stage has m $r \times r$ switches, and the third-stage has r $m \times n$ switches.

switches in the second stage, and $r(m - n + 1) \times 1$ switches in the third stage. In other words, only one input exists in each first-stage switch and one output in each third-stage switch. In this case, one second-stage $r \times r$ switch is enough to route the r inputs of r first-stage switches to the r outputs of r third-stage switches, thus completing the interconnect.

The reduction from $c(n - i, m - i, r)$ to $c(n - i - 1, m - i - 1, r)$ can be derived by a maximum matching algorithm. The matching algorithm selects disjoint signals from different input switches to different output switches. One second-stage switch is then used to route the selected signals. From Hall's theorem, the maximum matching and routing can always reduce the network to $c(n - i - 1, m - i - 1, r)$.

Conceptually, the routing problem can also be formulated as edge coloring on a bipartite graph $G(V_1, V_2, E)$ (31). The node sets V_1 and V_2 represent the switches in the input and output stages, respectively. An edge in E represents a two-pin net interconnect requirement between the corresponding input and output switches. In Reference (31), Chan and Schlag assigned colors to the edges of the bipartite graph. Edges of the same color are bundled into one group and the corresponding set of nets are routed by one switch in the second stage. The work of Reference (59) was then used to find a minimum edge coloring solution in $O(|E|\log n)$.

The three-stage Clos network can be folded into a two-stage network (Fig. 10) so that the inputs and outputs are mixed in the first stage. Thus, the corresponding bipartite graph $G(V_1, V_2, E)$ constructed above for edge coloring is also folded with V_1 and V_2 merged into one set.

To find the routing assignment, the folded edge coloring graph can be unfolded back to a bipartite graph using an Euler path search. The Euler path traverses every edge exactly once and defines the edge direction according to the direction of the traversal. We then recover the original bipartite graph by splitting the node set back into two sets V_1 and V_2 and unfold the edges such that all edges are directed from V_1 to V_2. We can find the minimum edge coloring solution of the unfolded bipartite graph and apply the solution back to the folded routing problem.

In practice, the first-level crossbar of the Clos network is replaced with FPGAs to save board space (Fig. 11). Routability is worse than an ideal Clos network. Even with a true Clos network, complete routability of multipin nets is not guaranteed, which is an important practical consideration because in microelectronic design, many multipin nets typically exist.

In an attempt to solve the multipin net and routability problem, we can introduce extra connections among FPIDs as shown in Fig. 12. However, extra FPID interconnections also incur extra delay. We can also expand the fanout width of FPGAs so that each FPGA I/O pin is connected to more than one FPIC (60, 61). The fanout width expansion improves routability without significant additional delay. The multiple appearances of I/O pins increase the probability that a signal connection can be made in a single stage, which is especially critical for multipin nets. However, the additional fanouts increase the needed pin count of FPICs. Thus, we need to find a balanced fanout distribution that reduces the interconnect delay with a minimal pin requirement.

A tree-structured network can simplify the mapping process for certain applications. In Reference (62), an example of a tree-structured network is illustrated for a Very Large Scale Simulator (VLSS). The VLSS tree structure has all logic components located at the leaves and interconnect switches at the internal nodes. The machine covers a capacity of eight million gates. Each branch is an 8-bit bus. The higher up the level of the tree, the less parallelism the signal distribution can achieve. Therefore, a partitioning process is designed to minimize the high level interconnect and maximize the parallel operation.

Interconnect Multiplexing

Time multiplexing is an effective method for tackling the scalability problem in interconnecting large designs. The time-sharing method can be extended from traditional bus organization (27, 62) to network sharing (63) and further to function block sharing (40).

Interconnect can be time shared as a bus (27, 62). If n communication lines exist between two FPGAs, they can be reduced to a single line by storing logical outputs in shift registers and time-multiplexing the communication in phases. Such a scheme was employed in the virtual wires logic emulation system (27), which is efficient because interconnects are normally capable of being clocked at much higher rates than the critical path of the rest of the system, and all logical wires are not simultaneously active. This scheme can greatly reduce the complexity of the interconnecting network or printed circuit board in a multi-FPGA system.

Li and Cheng (63) proposed that a dynamic network be viewed as overlapping L conventional FPICs together but sharing the same I/O pins. A dynamic routing architecture can increase the routability and shorten interconnect length. Each switching network is a full crossbar, which can be reconfigured to provide any connections among I/O pins. The select lines are used to activate only one switching network at a time; thus the I/O pins are dynamically connected according to the configuration of this active switching network. By dynamically reconfiguring the FPICs, L logic signals can time-share the same interconnect resources.

Memory Allocation

Interconnect schemes should also consider how memory is connected to the FPGAs. Although combining memory with logic in the same FPGA is the most desirable method for reducing routing congestion and signal delay, separate components can supply much larger capacity at higher density and lower price. Figure 13 demonstrates three different ways of allocating the memories in a Clos network (31, 64). The memory may be attached directly to a local FPGA (Fig. 13a), attached to the second-stage switches of the Clos network via a host interface (Fig. 13b), or attached to the first-stage switches of the Clos network (Fig. 13c). The first method provides good performance for local memory access. However, for the case of nonlocal memory access, the routability and delay are concerns. The second method is slower than the first method for local memory accesses but provides better routability. The third is the most flexible as the memory is attached to the network and the routability is high. However, every logic-to-memory communication must go through the second interconnect stage.

Bus Buffer Insertion

In FPGAs, signal propagation is inherently slow because of its programmable interconnect feature. However, the delay of long routing wires can be drastically reduced by buffer insertion. The principle at work is that by inserting buffers we can decouple capacitive effects of components and interconnect driven by the buffers and thereby improve RC delay.

Given a routing topology for a net and timing requirements for its sinks, an efficient optimal buffer insertion algorithm was proposed in Reference 65. Experimental results show dramatic improvement versus the unbuffered solution. Thus, it is advantageous to have abundant buffers in FPGAs. However, each possible buffer and its programmable switch adds capacitance to the wires, which in turn will contribute to delay. Thus, a balance point needs to be identified to trade off between the additional delay and capacitance of the buffers versus the improvement they can provide.

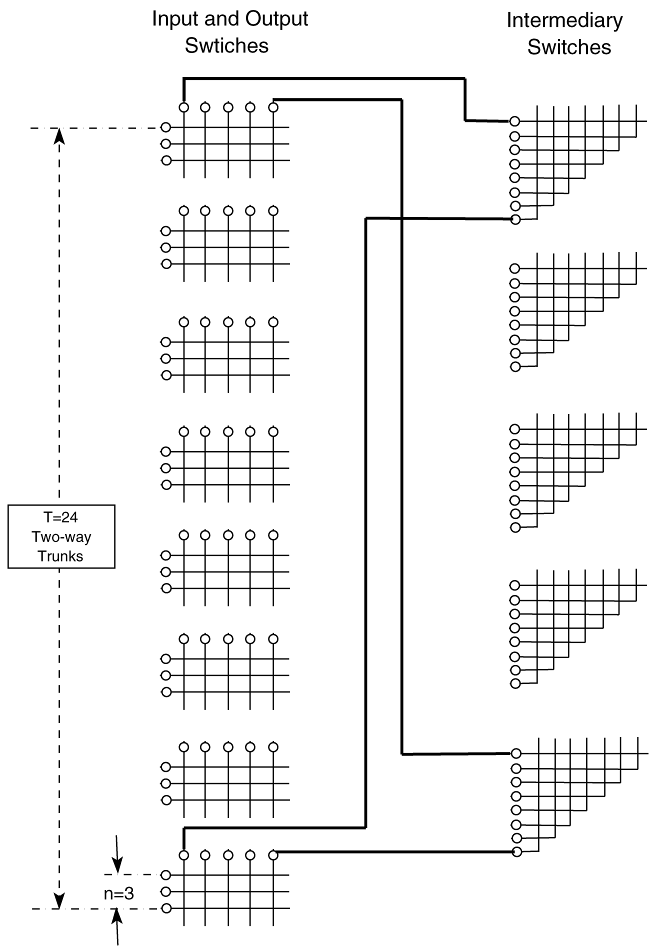


Figure 10. Folded Clos network. The three-stage Clos network is folded into a two-stage network so that the inputs and outputs are mixed in the first stage.

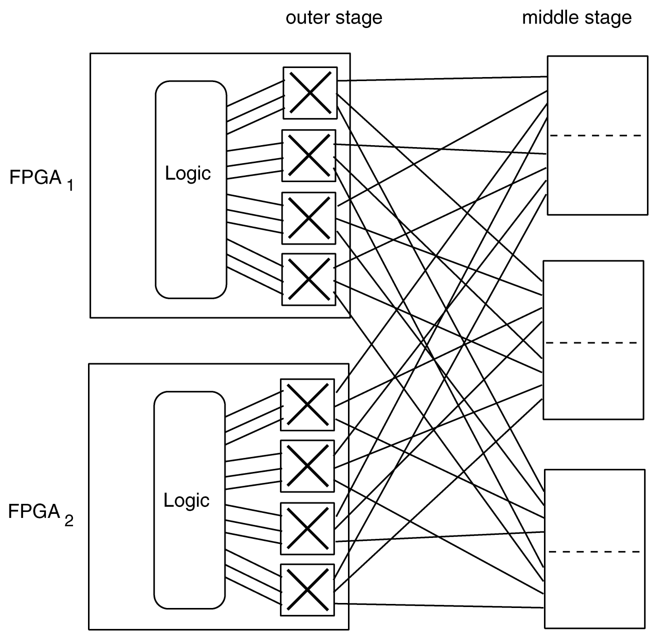


Figure 11. Variations of the Clos network. The first level crossbar of the Clos network is replaced with FPGAs to save board space. Routability is worse than an ideal Clos network.

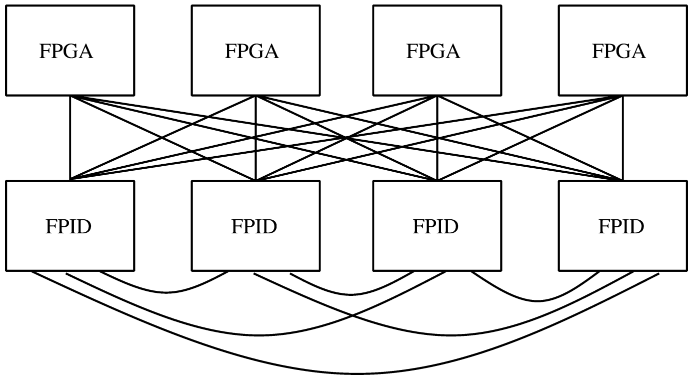


Figure 12. Variations of Clos network. The fanout width of FPGAs is expanded so that each FPGA I/O pin is connected to more than one FPIC. The fanout width expansion improves routability without significant additional delay.

For a multisourced bus, the problem of buffer insertion becomes more complicated, because the optimization for one source may sacrifice the delay of others. Furthermore, the direction of the buffer needs to be arbitrated by a controller. Instead of using such a controller, a novel approach is to use a patented open collector bus repeater (66). When idle, the two ends of the repeater are set to high. When the repeater senses the pull-down action on one side, it presents the signal on the other side until the pull-down action is released from the originated signal. The bus repeater eliminates the need for a direction control signal, resulting in a simpler design and better use of resources.

System Decomposition

To decompose a system into multiple devices, Yeh et al. (67) proposed an algorithm based on the relationship between uniform multi-commodity flow and min-cut partitioning. Yeh et al. construct a flow network wherein each net initially corresponded an edge with flow cost one. Two random modules in the network were chosen and the shortest path (i.e., path with lowest cost) between them was computed. A constant $\Delta < 1$ was added to the flow for each net in the shortest path, and the cost for every net in the path was incremented. Adjusting the cost penalizes paths through congested areas and forces alternative shortest paths. This random shortest path computation is repeated until every path between the chosen pair of modules passes through at least one "saturated" net. The set of saturated nets induces a multi-way partitioning in which two modules belong to the same cluster if and only if there is a path of unsaturated nets between them.

For each of these clusters, the *flux* (defined as the cut-size between the cluster and its complement, divided by the size of the cluster) is computed and the clusters are sorted based on their flux value. Yeh et al. began with a single cluster equal to the entire netlist, and then peeled off the clusters with lowest flux. This approach was attractive because the saturated nets are good candidates to be cut in a partitioning solution. As peeled clusters can be very small, a second phase may be used to make the multi-way partitioning more balanced. This approach, with its subsequent speedup by Yeh (68), is well-suited for large-scale multi-way partitioning instances.

The system prototyping phase may also explore netlist transformations such as logic replication and retiming to minimize cut size (I/O usage) or system cycle time. Such transformations are needed as inter-device delays can be relatively large and because devices are often I/O-limited. In Reference (69), Liu et al. proposed a partitioning algorithm that permits logic replication to minimize both cut size and clock cycle of sequential circuits. Given a netlist $G = (V, E)$, their approach chooses two modules as seeds s and t, then constructs a "replication graph" that is twice the size of the original circuit. This graph has the special property that a type of directed minimum cut yields the replication cut (i.e., a decomposition of V into S, T, and $R = V - S - T$ where $s \in S$, $t \in T$ and R is the replicated logic) that is optimal. A directed version of the Fiduccia-Mattheyses algorithm is used to find a heuristic directed minimum cut in the replication graph. Cong et al. (70) present an efficient algorithm for the performance-driven multi-way circuit partitioning problem that considers the different local and global interconnect delay introduced by the partitioning.

Alpert and Kahng (71) survey the FPGA partitioning literature in the context of major graph partitioning paradigms. The current partitioning problems are 1) low usage rate of FPGA gate capacity because I/O pin limit, 2) low clock rate because of interconnect delay between multiple FPGAs and 3) long CPU time for the mapping process.

System Planning and Design Changes

For a given system decomposition to be implemented on a multi-FPGA prototyping architecture, all connections within each device and between devices must be routable. Chan et al. (72) invoke much literature on routability prediction in gate arrays, as well as theoretical concepts, such as the Rent parameter, to obtain a fast routability estimate for arbitrary netlists and FPGA architectures. Their method ascribes one of three levels of routable (easily routable, marginally routable, or unroutable) to a netlist based on various parameters. Specifically, combining a wirelength estimator due to Feuer, the average number of pins-per-cell, and the estimated Rent parameter yields a relatively accurate routability predictor. The utility of these parameters is contrasted with that of other criteria such as El Gamal's channel width requirement (73) or the

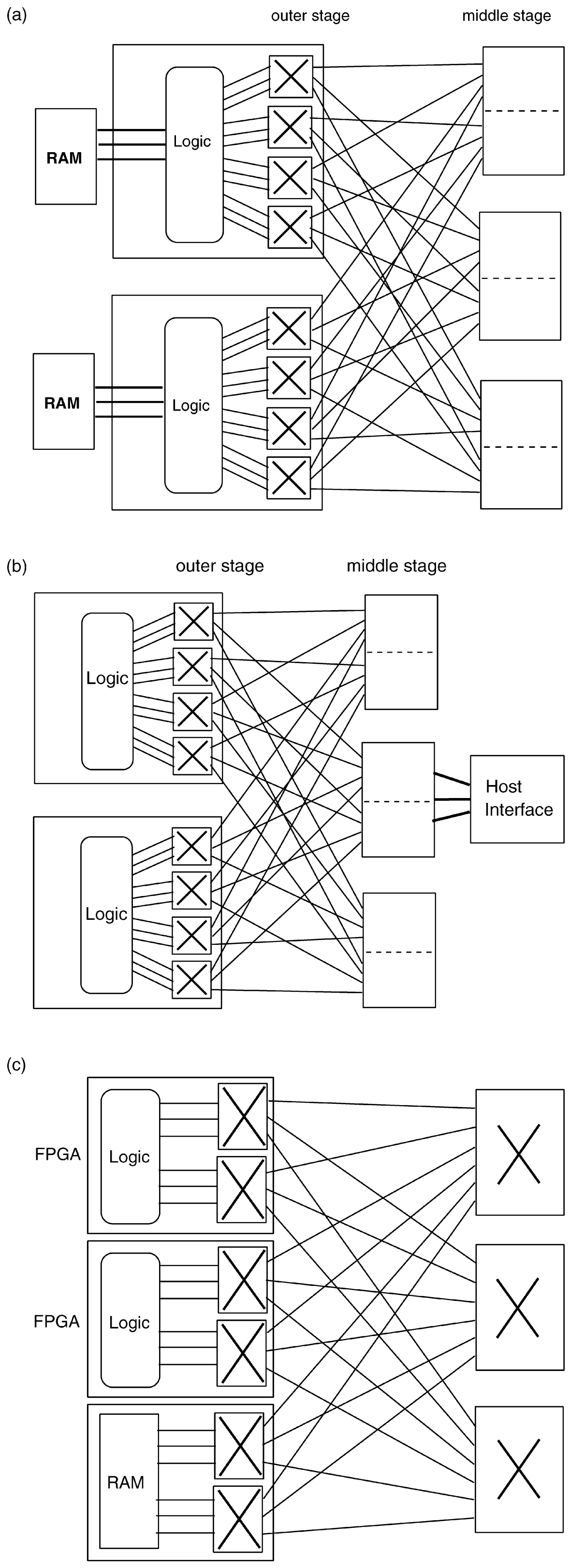


Figure 13. Memory organization, (a) Memory is attached directly to a local FPGA. (b) Memory is attached to the second-stage switches of the Clos network via a host interface, (c) Memory is attached to the first-stage switches of the Clos network.

average pins-per-net ratio.

In addition to routability, connections must also meet system timing constraints. Selvidge et al. (74) extend the original virtual wires (27) concept in their TIERS (Topology-IndEpendent Routing and Scheduling) approach. The problem formulation assumes that an assignment from a multiple-FPGA partitioning (i.e., a design graph) to a target topology graph has already been made. The objective is to assign "links" (i.e., signal nets) to channels between devices; as with the Virtual Wires concept, specific timeslices for a channel can be assigned to multiple links as long as no two links need to transmit signals at the same time. The TIERS algorithm uses a greedy method to order the links and then routes each link in the scheduled order while reserving channel resources; factors of up to 2.5 improvement in system cycle time are achieved.

Chang, et al. (75) address the combined issues of routability and system timing by applying layout-driven logic resynthesis techniques. For a given wire that cannot be routed, "alternative wires" and alternative functions are identified, such that the given unroutable wire can be removed from the circuit and replaced with a new wire (or wires) or new logic without affecting functionality. Cheng et al. estimate that between 30% and 50% of wires have so-called "triple-wire alternatives" (i.e., replacements consisting of three or fewer wires). Their method first routes the wires that do not have any alternatives then replaces any unroutable wire with available alternatives. System timing can be improved by replacing long wires with shorter alternatives.

CONCLUSION

Reconfigurable computing offers a middle ground between software-based systems and ASIC implementations, and is often able to combine important benefits of both. Implementations are able to avoid overheads such as unnecessary data transfers, decoding and control mandatory in microprocessors, and designs can be optimized on a basis specific to an application, a problem instance or even an execution. Using this technology, it is possible to achieve size, performance, cost, or power improvements over more conventional computing technologies.

ACKNOWLEDGMENTS

The authors would like to thank Y M. Lam for his help in preparing this manuscript and Prof. Wayne Luk (Imperial College) for his proofreading of this article.

BIBLIOGRAPHY

1. Estrin, G. Reconfigurable Computer Origins: The UCLA Fixed-plus-variable (F+V) Structure computer. *IEEE Ann. Hist. Comput.* 2002, **24** (4), pp 3–9.
2. Compton, K.; Hauck, S. The roles of FPGAs in Reprogrammable Systems. Proc. IEEE 1998, **86**, (4), pp 615–639.
3. Bondalapati, K.; Prasanna, V. Reconfigurable Computing System. *Proc. IEEE* 2002, 90 (7), pp 1201–1217.
4. Compton, K.; Hauck, S. Reconfigurable Computing: A Survey of System and Software. *ACM Comput.Surv.* 2002, **34** (2), pp 171–210.
5. Todman, T.; Constantinides, G.; Wilton, S.; Mencer, O.; Luk, W.; Cheung, P. Reconfigurable Computing: architectures and design methods. *IEE Proc. Comput. Digit. Tech.* 2000, **152**, (2), pp 193–205.
6. DeHon, A. The Density Advantage of Configurable Computing. *IEEE. Computer* 2000, **33** (4), pp 41–49.
7. Tsoi, K. H.; Lee, K. H.; Leong, P.H.W. A Massively Parallel RC4 Key Search Engine; *Proc. of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines FCCM*; Washington, DC, IEEE Computer Society, 2002, pp 13–21.
8. Liang, J.; Tessier, R.; Goecket, D. *A Dynamically-reconfigurable, Power-efficient Turbo Decoder*; 2004, pp 91–100.
9. Betz, V.; Rose, J.; Marquardt, A. Eds. *Architecture and CAD for Deep-Submicron FPGAs*; Kluwer Academic Publisher: Dordrecht, the Netherlands, 1999.
10. Xilinx.http://www.xilinx.com (accessed 2006).
11. Altera.http://www.altera.com (accessed 2006).
12. Actel.http://www.actel.com (accessed 2006).
13. Ahmed, E.; Rose, J. The Effect of LUT and Cluster Size on Deep-submicron FPGA Performance and Density *Proc. of the 2000 ACM / SIGDA Eighth International Symposium on Field Programmable Gate Arrays* ACM Press: New York, 2000; pp 3–12.
14. Goldstein, S. C.; Schmit, H.; Budiu, M.; Cadambi, S.; Moe, M.; Taylor, R. R. Piperench: A Reconfigurable Architecture and Compiler. *Computer* 2000, **33** (4), pp 70–77.
15. Ebeling, C.; Cronquist, D. C.; Franklin, P. Rapid - Reconfigurable Pipelined Datapath; *Proc. of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*; London, UK, Springer-Verlag: l996, pp 126–135.
16. M2000. FPC06: http://www.m2000.fr (accessed 2006).
17. eASIC. http://www.easic.com (accessed 2006).
18. Wilton, S.; Kafafi, N.; Wu, J.; Bozman, K.; Aken'Oven, V.; Saleh, R. Design Considerations for Soft Embedded Programmable Logic Core. *IEEE Solid Circuits* 2005, **40** (2), pp 485–497
19. Moll, L.; Vuillemn, J.; Boucard, P.; High-energy Physics on DECPeRLe-1 Programmable Active Memory; *Proc. of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays*; New York, (FPGA 95), ACM Press: 1995, pp 47–52.
20. Hoang, D. T. Searching Genetic Database on Splash, *IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1993, pp 185–191.
21. Ting, L.-K.; Woods, R.; Cowan, C. F. N. Virtex FPGA Implementation of a Pipelined Adaptive LMS Predictor for Electronic Support Measure Receivers. *IEEE Trans. VLSI Syst*. 2005, **13** (1), pp 86–95.
22. Chang, C.; Wawrzynek, J.; Brodersen, R. W. BEE2: A high-end Reconfigurable Computing System. *IEEE Des. Test* **22** (2), pp 114–125.
23. Shand, M.; Vuillemin, J.; Fast Implementations of RSA Cryptography: *Proc. 11th Symposium on Computer Arithmetic*, 1993, pp 252–259.
24. Zhang, G. L.; Leong, P. H. W.; Ho, C. H.; Tsoi, K. H.; Cheung, C. C. C.; Lee, D. -U.; Cheung, R. C. C.; Luk, W. Reconfigurable

Acceleration for Monte Carlo based financial simulation: *Proc. International Conference on Field Programmable Technology (ICFPT)*. 2005, pp 215–222.

25. Vuillemin, J.; Patrice, B.; Didier, R.; Shand, M.; Herve, T.; Philippe, B. Programmable Active Memories: Reconfigurable System Come of Age. *IEEE Trans. VLSI Syst*. 1996, **4** (1), pp 56–59.
26. Butts, M.; Batcheller, J.; Varghese, J.; An Efficient Logic Emulation System. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst*. 1993, pp 171–173.
27. Babb, J.; Tessier, R.; Dahl, M.; Hanono, S. Z.; Hoki, D. M.; Agarwal, A. Logic Emulation with Virtual Wires. *IEEE Trans. Computer-Aided Design* 1997, **16** (6), p 609.
28. de Souza, L.; Ryan, P.; Crawford, J.; Wong, K.; Zyner, G.; McDermott, T. Prototyping for the Concurrent Development of an IEEE 802.11 Wireless LAN Chaipset: *Proc. International Conference on Field-Programmable Logic and its Applications* LNCS 2778, Springer, 2003, pp 51–60.
29. Cadence. http://www.cadence.com/datasheets/lncisivePalladiumllds.pdf (accessed 2006). Palladium Data Sheet, 2005, pp 1–8.
30. Bertin, P.; Roncin, D.; Vuillemin, J. Introduction to Programmable Active Memories. *DEC Memo 3* 1989, pp 1–9.
31. Chan, P. K.; Schlag, M. D. F. Architectural Tradeoffs in Field-programmable Devices Based Computing System; *EEE Workshop on FPGAs for Custom Computing Machines*; l993, pp 152–161
32. Mohsen, A. Programmable Interconnects Speed System Verification. *IEEE Circuits Devices Mag*, 1993, **9** (3), pp 37–42.
33. Slimane-Kadi, M.; Brasen, D.; Saucier, G. A Fast-FPGA Prototyping System That Uses Inexpensive High-performance FPIC. *ACM Int. Workshop on FPGAs*, Berkeley, CA, 1994, pp 1.3. 1–11.
34. Arnold, J. M.; Buell, D. A.; Davis, E. G. SPLASH 2; *4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 9Z)*; 1992, pp 316–322.
35. Cray. http://www.cray.com/downloads/Cray_XDl_Datasheet.pdf (accessed 2006). Cray XDl Datasheet.
36. SRC Computers. http://www.srccomp.com (accessed 2006).
37. SGI. http://www.sgi.com/products/rasc/ (accessed 2006).
38. Villasenor, J.; Mangione-Smith, W. H. Configurable Computing. *Scientif. Amer.* 1997, pp 67–71.
39. Becker, J.; Hubner, M. Run-time Reconfigurability and Other Future Trendy; *Proc. of the 19th Annual Symposium on Integrated Circuits and Systems Design (SBCCI 06)*; ACM Press: New York, 2006, pp 9–11.
40. Bhat, N. B.; Chaudhary, K.; Kuh, E. S. Performance-oriented Fully Routable Dynamic Architecture for a Field-programmable Logic Device. *Memorandum No. UCB/ERL M93/42, Electronics Research Lab., College of Engineering, UC Berkeley*, 1993, pp. 1–21.
41. NEC Electronics. http://www.necel.com/drp/in/index.html (accessed 2006).
42. Wirthlin, M. J.; Hutchings, B. L. A Dynamic Instruction Set Computing; *Proc. of the IEEE Symposium on FPGA's for Computing Machines* Washington, DC, IEEE Computer Society, 1995, pp 99–107.
43. Stretch, Inc. http://www.stretchinc.com (accessed 2006).
44. Steiger, C.; Walder, H.; Platzner, M. Operting System for Reconfigurable Embedded Platforms: Online Scheduling of Real-time Task. *IEEE Trans. Comput.* 2004, **53** (11), pp 1393–1407.
45. Schmit, H. Incremental Reconfiguration for Pipelined Applications; *Proc. of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines* Washington, DC, IEEE Computer Society, 1997, pp 47–55.
46. Young, S.; Alfke, P.; Fewer, C.; McMillan, S.; Blodget, B.; Levi, D. A High I/O Reconfigurable Crossbar Switch; *Proc. of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 03)*; Washington, DC, IEEE Computer Society, 2003, pp 3–10.
47. Guccione, S.; Levi, D.; Sundarajan, P. A Java-based Interface for Reconfigurable Computing; *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*; September 1999. citeseer.ist.psu.edu/681383.html.
48. Luk, W.; Page, I. *Compiling Occam into FPGAs*. *FPGAs*. Abingdon, EE &CS books: 1991; pp 271–283.
49. Page, I. Constructing Hardware/Software System From a Single Description. *VLSI Signal Processing*, 1996, **12**,pp 87–107.
50. Celoxica. http://www.celoxica.com (accessed 2006).
51. Mentor Graphics. http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/index.cfm (accessed 2006).
52. Mathworks, Inc. http://www.mathworks.com (accessed 2006).
53. Haldar, M.; Nayak, A.; Choudhary, A.; Banerjee, P. A System for Synthesizing Optimized FPGA Hardware from MATLAB; *Proc. of the 2001 IEEE/ACM International Conference on Computer-aided Design (ICCAD)* Piscataway, NJ, IEEE Press: 2001, pp 314–319.
54. Lau, D.; Pritchard, O.; Molson, P. Automated Generation of Hardware Accelerators with Direct Memory Access from ANSI/ISO Standard C Functions, *Proc. of the 14th Annual IEEE symposium on Field-Programmable Custom Computing Machines (FCCM'06)*, Washington, DC, IEEE Computer Society, 2006, pp 45–56.
55. Isshiki, T.; Dai, W. M. High-level Bit-serial Datapath Synthesis for Multi-FPGA System; *International Workshop on FPGAs*; 1995, pp 167–174.
56. Leong, M. P.; Leong, P. H. W. A Variable-Radix Digit-serial Design Methodology and Application to the Discrete Cosine Transform. *IEEE Trans. VLSI System* 2003, **11** (1), pp 90–104.
57. Clos, C. A Study of Nonblocking Switching Networks. *Bell System Tech. J.* 1953, **32**,pp 406–424.
58. Benes, V.E. *Mathematical Theory of Connecting Networks and Telephone Traffic*; Academic Press: New York, 1965.
59. Cole, R.; Hopcroft, J. On Edge Coloring Bipartite Graphs. *SIAM J.Computing* 1982, **11**,pp 540–546.
60. Richards, G. W.; Hwang, F. K.A Two-stage Rearrangeable Broadcast Switching Network. *IEEE Trans. Commun.* 1985, COM- **33**(10), pp 1025–1035.
61. I-Cube, Using FPID Devies in FPGA-based Prototyping. *Application Note* 1994, pp 1–11.
62. Wei, Y. C.; Cheng, C. K.; Wurman, Z. Multiple-level Partitioning: An Application to the Very Large-Scale HardWare Simulator. *IEEE J. Solid State Circuits* 1991, pp 706–716.
63. Li, J.; Cheng, C. K.. Routability Improvement Using Dynamic Interconnect Architecture. *IEEE FPGAs for Custom Computing Machines* 1995, Apr, pp 13.2.–7.
64. Chan, P. K.; Schlag, M. D. F.; Martin, M. BORG: A Reconfigurable Prototyping Board Using Field-programmable Gate Arrays. *Int. Workshop on FPGA*; Berkeley CA, 1992, pp 47–51.

65. Lillis, J.; Cheng, C. K.; Lin, T. T. Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model. *IEEE/ACM Int. Conf. on Computer-Aided Design*; 1995, pp 138–143.

66. Hsieh, W. J.; Jenq, Y. C.; Horng, C. S.; Lofstrom, K. Input/output I/O Bidirectional Buffer for Interfacing I/O Parts of a Field Programmable Interconnection Device with Array Ports of a Cross-point Switch. *US Patent no 5,428,800*, 1992, pp 1–13.

67. Yeh, C. W.; Cheng, C. K.; Lin, T.T. A Probabilistic Multicommodity-flow Solution to Circuit Clustering Problems. *IEEE Int. Conf. on Computer-Aided Design*; 1992, pp 428–431.

68. Yeh, C. W. On the Acceleration of Flow-oriented Circuit Clustering. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*. 1995, pp 1305–1308.

69. Liu, L. T.; Kuo, M. T.; Cheng, C. K.; Hu, T.C. Performance-driven Partitioning Using a Replication Graph Approach. *ACM/IEEE Design Automation Conf*; June 1995, pp 206–210.

70. Cong, J.; Lim, S. K.; Wu, C. Performance driven Multi-level and Multiway Partitioning with Retiming; *Proc. of the 37th Conference on Design Automation (DAC 00)*; New York, ACM press: 2000, pp 274–279.

71. Alpert, C. J.; Kahng, A.B. Recent Directions in Netlist Partitioning: A Survey. *Integration, The VLSI.* 1995, August,pp 1–81.

72. Chan, P. K.; Schlag, M. D. F.; Zien, J. Y. On routability Prediction for Field-programmable Gate Arrays; *IEEE Design Automation Conf.*; Dallas, 1993, pp 326–330.

73. El Gamal, A. Two-dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits. *IEEE Trans. CAS.* 1981, **28** (2), pp 127–138.

74. Selvide, C.; Agarwal, A.; Dahl, M.; Babb, J. TIERS: Topology Independent Pipelined Routing and Scheduling; *Int. Symp. on FPGA*; 1995, pp 25–31.

75. Chang, S. C.; Cheng, K. T.; Woo, N. S.; Marek-Sadowska, M. Layout Driven Logic Synthesis for FPGA; *Proc. ACM/IEEE Design Automation Conference.* 1994, pp 308–313.

76. Leong, M. P. *FPGA Design Methodologies for High Performance Applications.* PhD dissertation, The Chinese University of Hong Kong, 2001.

CHENG CHUNG-KUAN
ANDREW B. KAHNG
PHILIP H.W. LEONG
Dept. of Computer Science and Engineering, University of California, La Jolla, California
Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong