CSE-670 TERM REPORT

A SURVEY OF LOGIC BLOCK ARCHITECTURES

FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

SUBMITTED BY MUSTAFA IMRAN ALI
SUBMITTED TO DR.M.E.S.ELRABAA

TABLE OF CONTENTS

TABLE OF FIGURES

A SURVEY OF LOGIC BLOCK ARCHITECTURES

FOR DIGITAL SIGNAL PROCESSING APPLICATIONS

ABSTRACT

The lower efficiency of FPGAs compared with custom ASICs can be improved upon if the characteristics of the target application can be exploited in the design of FPGA architecture, and in particular, an efficient logic block. The increasing use of FPGAs for a particular application domain, such as Digital Signal Processing, justifies the application domain tuning of the FPGA architecture. By trading off the general purpose nature of an FPGA with application specific optimizations, we can improve the cost, performance and power efficiencies. In this report, the various techniques used to design optimized logic blocks are presented and compared. The best suitability of each class of optimization approach is then determined with respect to the needs of different applications.

1. INTRODUCTION

Although FPGAs are a cost-efficient alternative for both ASICs and general purpose processors, they still result in designs which are more than an order of magnitude costly and slower than their equivalents implemented in dedicated logic. This efficiency gap makes FPGAs less suitable for high-volume cost-sensitive applications (e.g. embedded systems). What makes FPGAs so attractive is the flexibility they offer. However, the same flexibility, or precisely the way how it is obtained (e.g. look-up table based logic, huge interconnect network), results in a large cost-efficiency gap between FPGAs and ASICs. Finding a good balance between flexibility and efficiency (in terms of area, performance and power) in reconfigurable logic devices is not trivial. This can be seen from the fact that reconfigurable logic architectures have hardly changed over the years. The fabric of today's FPGAs, although enhanced with some extra features, still very much resembles the one used in the first FPGAs; paradoxically, interconnect is made even richer. Therefore, in spite of mature 'know-how' and technological advances which only decreases the absolute cost of a single transistor, FPGA designs still suffer from their high intrinsic cost. This cost, although accepted for some applications, can be a crucial limiting factor for the others [1].

The logic block of an FPGA, which is the focus of this report, is the basic computing element and its design is heavily influenced by the type of computations that are to be efficiently supported. Due to their varying requirements, the computations in any digital system can be broadly classified as either control or data oriented. The amount of each computations and their exact structure can be identified to design an efficient logic block that is optimized for a particular application domain. This can be achieved by exploiting the idea of application-domain tuning. This is based on the observation that most FPGAs, although made general-purpose, are often used for specific classes (domains) of applications only. The identification of characteristic properties of designs from a given application class leads to translating them into architectural improvements. The idea is to analyze a set of representative digital signal processing benchmarks and identify the characteristics of computations required and then applying this knowledge in the design of an efficient architecture. The architecture resulting from such an exercise will be superior to any general purpose architecture given that digital signal processing applications are being implemented. The choice of digital signal processing domain was made for this report as it typifies a computationally intensive domain and a study for DSP optimization can serve as a building block for understanding how efficient

reconfigurable systems can be designed to accelerate a broad category of computationally intensive application domains. This report is organized as follows. In section 2, the problem of computation differences in datapath and control logic is discussed. Section 3 introduces the concept of application domain characterization and how it can be carried out. In section 4, a classification of various datapath oriented reconfigurable architectures is presented. In Section 5, details of a few logic block designs that specifically target DSP are detailed. Section 6, presents a comparison of the various approaches and finally in Section 7 conclusions are presented.

2. PROBLEM DEFINITION

DSP applications are usually considered as being dominated by arithmetic or datapath computations. However, a careful analysis of various DSP benchmarks indicates the presence of other types of computations too, e.g. bit-level manipulations or small pieces of random logic. Therefore, for an efficient implementation of DSP, both datapath and random logic type of functionality must be supported. In FPGAs, one of the problems which make this requirement difficult to realize is the different nature of these computations. For example, from the functionality point of view, datapath functions operate on coarser arguments than those which are usually processed by random logic. At the same time, however, the implementation of datapath functions, and in particular arithmetic functions, is usually realized by fine-grain elements, while the implementation of random logic mostly benefits from coarser granularity. The reason for this 'paradox' is the underlying computing structure of FPGAs, i.e. a LUT (look-up table)-based processing element. The LUT complexity (in terms of the number of its configuration bits) grows much faster with the increase of the bit-width of input arguments for arithmetic than for logic operations [1]. Moreover, the generation of consecutive output bits of arithmetic operations depends on the propagation of a carry signal. If this carry dependence is coded within the LUT, its size grows exponentially. Since a configuration memory is very costly (especially for embedded systems), the implementation of arithmetic operations in finer LUTs, and a serial carry propagation between them, is preferred [1]. On the other hand, because logic functions are usually implemented as multi-level nets of gates, and have relatively few inputs and outputs, such functions clearly benefit from the implementation in coarser (larger) LUTs. The reason is that such LUTs usually reduce the total logic-depth, and thus the path delay. In the DSP-optimized FPGAs, the logic block architecture should reflect the conflicting granularity requirements of datapath and random logic functions as described above.

3. APPLICATION DOMAIN TUNING

In conventional general-purpose FPGAs performance is traded for flexibility [1]. This allows postponing a decision on what precisely is to be mapped, of what complexity, and in what way to the very last moment of the design process. Moreover, it permits reusing the same piece of silicon for designs of completely different natures. Though attractive, such flexibility is not always required, and (because of the associated cost) sometimes even undesirable. Therefore, if a given application domain offers optimization opportunities, they should be exploited. The idea of the application-domain tuning proposed here allows to trade back the ultimate flexibility of FPGA devices for some reduction in their intrinsic cost. This is possible if there is some pre-knowledge on the type of functions which are to be implemented. A general characterization of such functions and identification of characteristic properties of a given application domain (class) allows to optimize FPGA fabrics. After such an optimization, an FPGA fabric is no longer general-purpose but domain-specific. In contrast to the approach taken in [10], architectures based on application-

domain tuning concept presented here offer much more flexibility than the custom reconfigurable logic that is tailored to a limited set of functions only. On the one hand, this increases the performance penalty, but, on the other hand, it lowers the risk (the mapped functions can still be exchanged with new ones from the same application domain). The implementation of functions with completely different characteristics is possible too, but at a higher implementation cost.

3.1 DSP APPLICATION-DOMAIN CHARACTERIZATION

Digital signal processing plays an essential role in many modern applications, and there is a strong need for DSP-specific tuning to meet requirements of such applications. This trend can be observed in the increasing number of publications describing different DSP-optimized FPGA architectures.

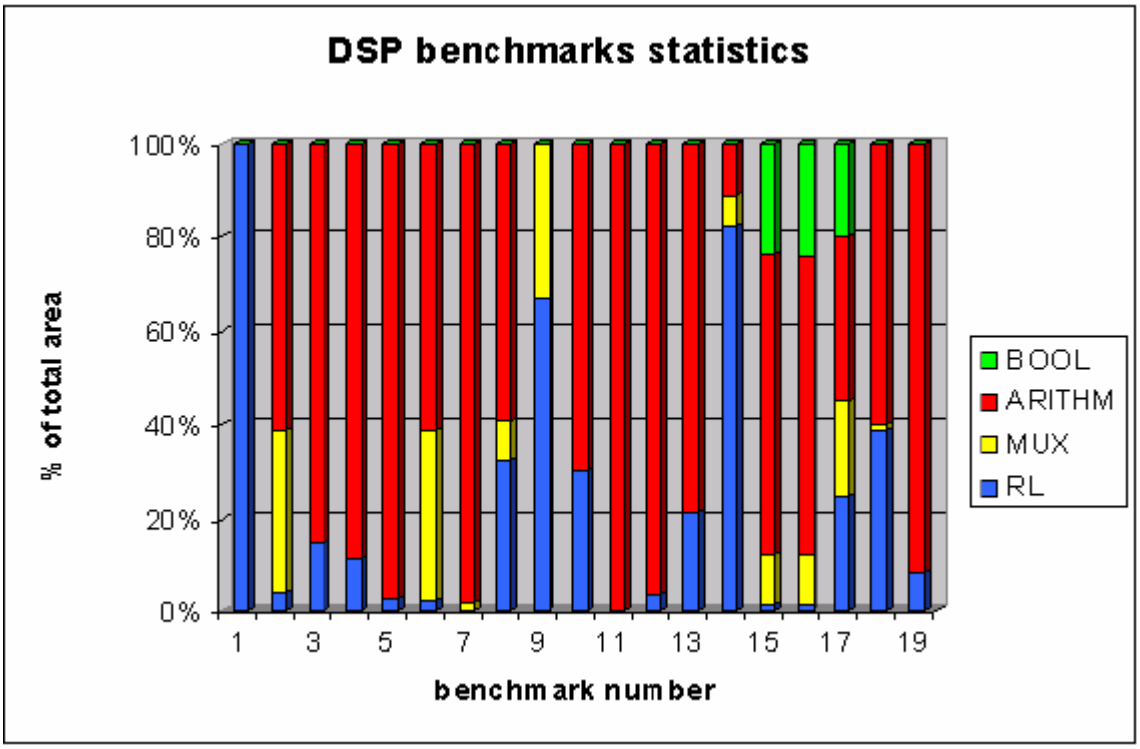

Figure 1: Statistics for 19 industrial DSP designs showing the type and amount of computations required in each design.

To perform an application-domain analysis for the DSP application domain, a representative set of DSP benchmarks are characterized in terms of the type of used computations and the frequency of their occurrence. The methodology used for this purpose is as follows. The authors [1] synthesized 19 industrial DSP designs of different complexity: from small ASUs (Application Specific Units) to large filters. Cadence BuildGates synthesis tool-set with AmbitWare datapath library was used. The latter allows extraction of various macro-blocks, and their preservation (if required) during the mapping process. In these experiments, the global synthesis parameters are set such that all datapath components with operands equal or larger than 4-bits and all multiplexers were treated as macro-blocks. Because a technology independent gate-level netlist generated in this way does not contain information on the area occupied by different components, the designs were mapped onto standard cells from a CMOS 0.13 µm library. For each design, a resource report was generated with the area information. All design components were classified in two main groups, i.e. datapath logic (implemented by macro-blocks) and random logic (implemented by simple gates). For an accurate

analysis of datapath functionality, distinguishing between arithmetic components, multiplexers and wide Boolean functions is required. In figure 1, the information on the properties of the mapped designs is presented in the form of a chart. The chart shows the percentage of area taken by random logic and datapath components. Figure 2 shows a random logic portion of the designs, and identifies the amount of sequential logic (flip-flops) which is present there. This is to make sure that the identified random logic is not confused with flip-flops.

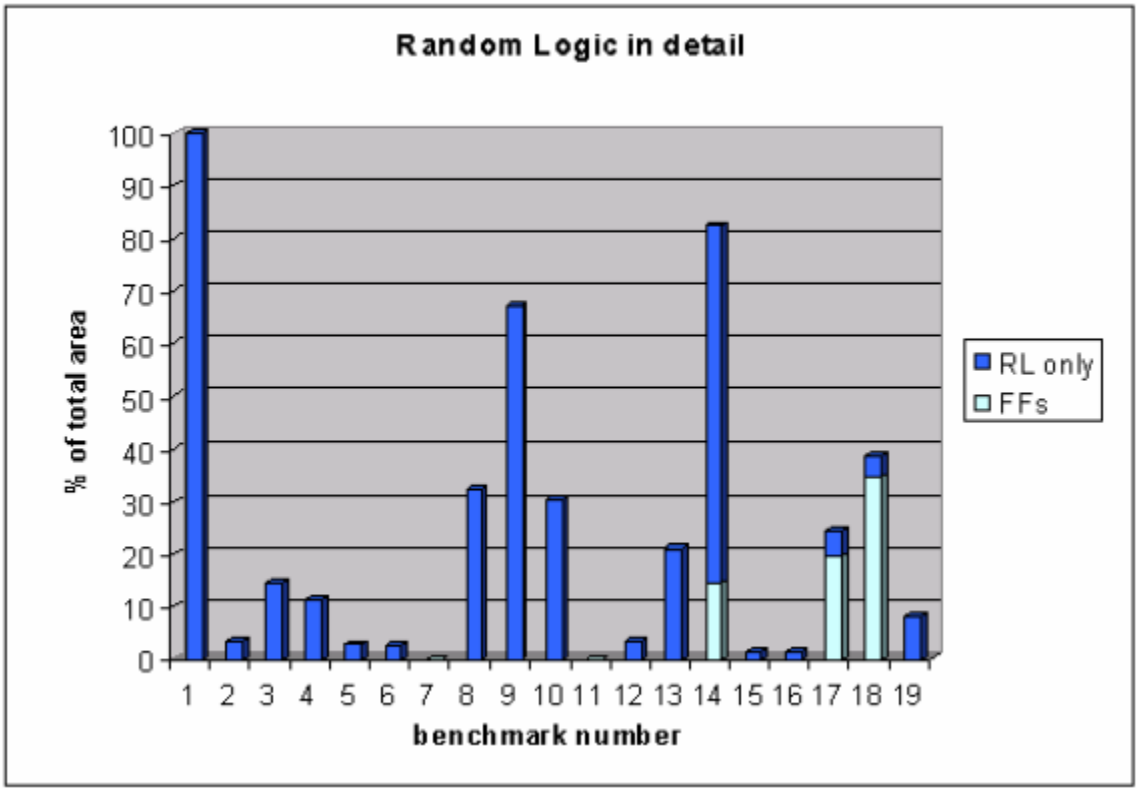

Figure 2: Combinatorial (pure RL) versus sequential (FFs) random logic in the mapped designs.

From the obtained results, the following conclusions can be drawn:

1. As expected, the datapath functionality, and in particular arithmetic, is dominant in DSP. The datapath functions have different bit-widths.

2. DSP designs heavily use multiplexers of various sizes. Thus, an efficient mapping of multiplexers should be supported.

3. DSP functions do contain random logic. The amount of random logic varies per design, but on average does not exceed 25% of the total design area. Random logic is predominantly combinatorial.

4. Some DSP designs use wide Boolean functions. This fact should be reflected in the logic block architecture.

4. A CLASSIFICATION OF DATAPATH ORIENTED RECONFIGURABLE ARCHITECTURES

This section provides a general survey on the field of datapath-oriented field-programmable architectures that are typically designed for arithmetic-intensive applications [9]. These datapath oriented architectures can be classified into four classes, which include the processor-based architectures, the static Arithmetic Logic Unit (ALU)-based architecture, the dynamic ALU-based architectures, and the LUT-based architectures. Each of these architectural classes is described in turn; and the section is concluded by a brief review of the various datapath-oriented features employed in the current state-of-the-art commercial FPGAs. Note that the first three architectural classes are built around arrays of small processors or ALUs, which are considerably more complex than LUTs. As a result, these devices usually have quite different routing demands and contain substantially different routing resources than the conventional FPGAs. The fourth class of architectures is more FPGA-like; each contains an array of LUT-based logic blocks and segmented routing resources. As a result, this class of devices is more closely related to the application domain characterization work presented in this report.

4.1 PROCESSOR-BASED ARCHITECTURES

The architectures of PADDI-1 [Chen92], PADDI-2 [Yeun93], RAW machine [Wain97], and REMARC [Taka98] all consist of an array of processors. These architectures can be said to be reconfigurable on a cycle-by-cycle basis since, whenever a processor executes a new instruction, the behavior of the processor is changed. This cycle-by-cycle reconfigurability is quite different from the conventional FPGAs whose logic blocks are configured only once - at the beginning of a computation process. In particular, PADDI-1, PADDI-2, and REMARC all use simple 16-bit wide processors that can perform addition, subtraction, and several logical operations in hardware. Each processor also contains a 16-bit wide register file (PADDI-1 and PADDI-2) or data memory (REMARC) for storing data. Each also has enough instruction memory to store a maximum of 8 (PADDI-1 and PADDI-2) or 32 (REMARC) instructions. Note that these instructions are stored in fully decoded forms so each instruction might take up to 32 to 53-bits of storage space. The instructions are executed in an order either as indicated by a global program counter (PADDI-1 and REMARC) or as specified in the next-program-counter field of each instruction (PADDI-2). The RAW machine is composed of an array of full-scale 32-bit wide processors, each containing a large instruction memory and data memory, as well as a register file. The ALU inside each processor can perform a variety of arithmetic and logical operations including hard-wired multiplication and division. The processor also contains a substantial amount of reconfigurable logic in the form of LUTs and flip-flops. Note that many specifics of the RAW machine including the number of instruction and data memory entries, the amount of registers in each register file, the exact structure of the ALU, and the size of the reconfigurable logic are variable architectural parameters.

The processors communicate with each other through global connection networks, which vary widely from architecture to architecture. In particular, the PADDI-1 device employs a crossbar network that connects two rows of four processors together. The connections are made in chunks of 16-bit wide buses. The PADDI-2 architecture is built upon the PADDI-1 architecture. Here eight processors are connected into a cluster using the crossbar network of PADDI-1. Sixteen clusters are then grouped into two rows of eight clusters. Between these two rows are several horizontal routing buses that run the full length of the row. The clusters are connected to the buses through their input or output pins. For better performance, each routing bus is broken into segments using programmable switches at pre-determined intervals. The buses are time-shared resources. To communicate, a processor has to use a set of pre-defined communication protocols to

claim a bus. Once a bus is claimed, the communication can be bidirectional - either to or from the initiating processor.

Each REMARC device contains 64 processors in an 8 by 8 array. There is only one 16- bit wide bus running horizontally or vertically in between every two rows or two columns of processors. The communication is again time-shared and uses a set of pre-defined protocols. Note that in this architecture, unlike the conventional FPGAs, a horizontal bus does not connect to a vertical bus.

Finally, the RAW machine has an elaborate routing architecture. Similar to REMARC, its processors are arranged in an array structure. There are a number of 32-bit wide buses running horizontally or vertically in between two rows or two columns of processors. Like conventional FPGAs, at the intersection of a horizontal and vertical routing channel, there is a switch block. Unlike conventional FPGAs, however, each switch block contains a set of instruction memory (controlled by the program counter of a nearby processor) whose content controls the cycle-by-cycle connectivity of the switch block. The switch block also can perform wormhole routing of packets generated by the processors using the addresses contained in the header of each packet.

4.2 STATIC ALU-BASED ARCHITECTURES

Unlike processor-based architectures, static ALU-based architectures, including Colt [Bitt96], DReAM [Also00], and PipeRench [Gold00], do not contain instruction memory, the program counter, and their associated control logic. Instead, the configuration of each ALU is directly controlled by the configuration memory. Nevertheless, ALU-based architectures still can be rapidly reconfigured since these architectures consume significantly less configuration memory than the traditional FPGAs.

A Colt logic block is called an IFU, which contains a 16-bit wide ALU and several pipeline registers. The device consists of 16 IFUs placed in a 4 by 4 array. Each IFU is connected to its immediate neighbors through a set of nearest neighbor interconnects. Two 16-bit wide inputs of each IFU located at the top row of the array and one 16-bit wide output of each IFU located at the bottom row of the array are connected together by a full crossbar (called the smart crossbar) which also provides partial connectivity to chip-level I/Os. A DReAM logic block is called a RPU, which contains two 8-bit wide ALUs and two banks of 8-bit wide memory. Four RPUs are grouped into a cluster. Within the cluster, RPUs communicate with each other through a set of 16-bit wide cluster-level local interconnects. Nine clusters in a 3 by 3 array form a DReAM device. The array is interconnected by a global routing network, which is similar in topology to a conventional FPGA global routing network. As in conventional FPGAs, the horizontal and vertical routing channels are connected together by switch blocks at their intersections. Unlike conventional FPGAs, however, the routing tracks are grouped into 16-bit wide buses; and the RPUs communicate across these buses through a set of pre-defined communication protocols. Note that besides the RPUs, each DReAM device also contains a global communication unit whose function and structure is beyond the scope of this work.

Each ALU-based logic block of PipeRench is called a stripe, which contains sixteen 8-bit wide ALUs and a set of registers. Stripes in a PipeRench device are vertically stacked; and the physical routing network of the device only provides connectivity between two adjacent stripes. Communication across distant stripes is achieved through rapid reconfiguration and by storing data in the internal registers of a stripe. The technique, called virtual global connection, is described in more detail in [Gold00]. Although essential to the structure of PipeRench, the technique cannot be readily applied to the traditional FPGAs and is not described in detail here.

4.3 DYNAMIC ALU-BASED ARCHITECTURES

Like the static ALU-based devices, the RaPiD [Ebel96] and Chess [Mars99] architectures contain only ALU-based logic blocks and no instruction memory. These ALUs, however, not only can be configured by configuration memory but also by data from the computation process itself. This extra level of flexibility increases the functionality of the architectures at the expense of increased architectural complexity and hardware cost. In particular, each RaPiD device is composed of identical functional units, which consist of groups of 16-bit wide datapath components including ALUs, registers, RAM blocks, and integer multipliers. The functional units are linearly placed in a row and connected to a set of routing tracks through programmable switches. These tracks are grouped into 16-bit wide buses and run horizontally across the full length of the row. To increase speed, each track is broken into a series of wire segments, which are interconnected by programmable switches. Each Chess device consists of a set of 4-bit wide ALUs placed in an array, which is interspersed by RAM blocks. The ALUs and RAM blocks are connected by an FPGA-like global routing network. The ALUs are also directly connected to their immediate neighbors by a set of nearest neighbor interconnects.

4.4 LUT-BASED ARCHITECTURES

Architectures such as Garp [Haus97], the data path FPGA (DP-FPGA)[21], Computationally Field Programmable Architecture (CFPA)[6], and the more recent Multi-bit FPGA (MB-FPGA)[9] are LUT-based architectures intended to support data path computations. A Garp device is designed as a reconfigurable functional unit of a MIPs processor. The architecture consists of an array of 32 rows by 24 columns of logic blocks. Each logic block contains two LUTs that are controlled by a single set of configuration memory. Each LUT has four inputs and is connected by a set of fast carry connections to other LUTs that are on the same row. The global routing network of Garp is similar to the global routing network of a conventional FPGA in topology. However, unlike the conventional routing network where wire segments are connected together by routing switches, wire segments of Garp remain unconnected to each other. The exclusion of routing switches significantly reduces the configuration memory required to control a Garp device and can lead to faster reconfiguration. It also, however, severely limits the possible applications of the Garp architecture. The latter three architectures are closely related to the contemporary FPGA as well as to the DSP optimized logic block architectures to be presented in the next section. They are discussed in somewhat detail here.

4.4.1 DATAPATH FPGA (DP-FPGA)

The overall structure of the DP-FPGA is shown in figure 3. It consists of three high-level blocks including the memory block, the control block, and the datapath block.

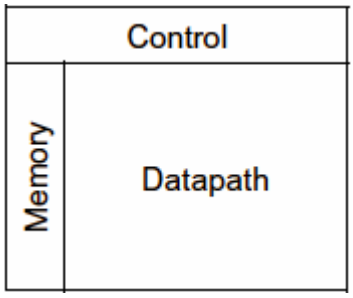

Figure 3: Overall structure of DP-FPGA

The datapath block of DP-FPGA is one of the first FPGA architectures that use the technique of configuration memory sharing (CMS) to create both the CMS routing resources and the CMS logic blocks. The technique creates CMS resources by sharing a single set of configuration memory among several programmable resources. By sharing, the amount of configuration memory that is needed to control the programmable resources is reduced and consequently, the implementation area of datapath applications, which contains a large amount of identical bit-slices, is minimized. The study demonstrates that there can be significant savings in logic block area when CMS logic blocks are used instead of conventional logic blocks for implementing datapath circuits.

The main building blocks of a DP-FPGA logic block are arithmetic LUTs. Structurally, these LUTs are more complex than the conventional LUTs used in the conventional architecture. An arithmetic LUT, shown in figure 4, consists of n inputs, two outputs, two conventional LUTs each with n-1 inputs, two two-input multiplexers and an SRAM cell. The LUT has two modes of operations, the normal mode and the arithmetic mode, which are controlled by the SRAM cell. When the SRAM cell is set to be one, the arithmetic LUT is in the normal mode of operation. In this mode, it behaves as a conventional LUT with n-inputs and one output. The LUT output is presented on the output P shown in figure 4 and the output G is ignored. When the SRAM cell is set to be zero, the LUT is in the arithmetic mode of operation. In this mode, one conventional LUT in the arithmetic LUT is used to generate the output at P, which is used as a propagate signal of a carry look ahead adder. The other conventional LUT in the arithmetic LUT is used to generate the output at G, which is used as a generate signal of a carry look ahead adder.

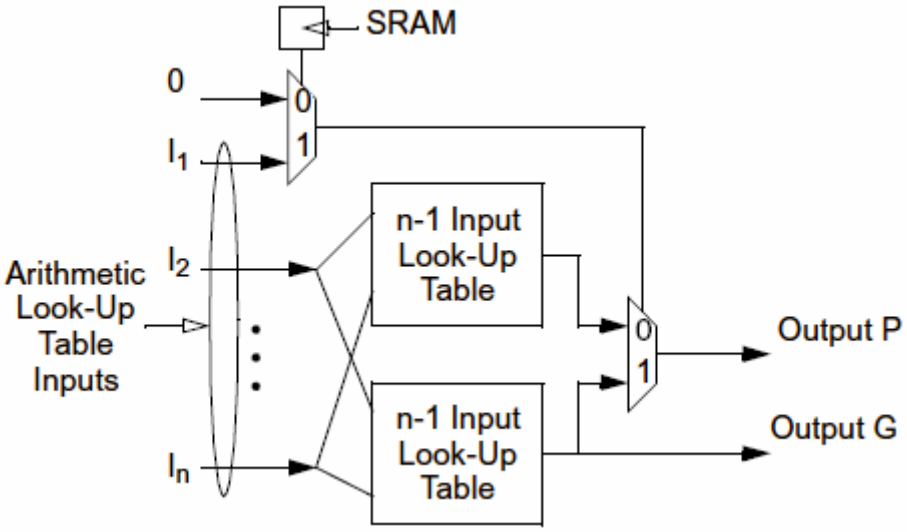

Figure 4: Arithmetic look-up table based logic block of DP-FPGA

The internal structure of the logic block is shown in Figure 5. It consists of four arithmetic LUTs, one carry block, four flip-flops, and four two-input multiplexers. All four arithmetic LUTs have the same number of inputs. Each LUT input is connected to a unique data input of the logic block. All four LUTs share a single set of configuration memory and are identically configured at all times. The SRAM cell that controls the operation mode of the arithmetic LUTs is also shared across all four LUTs. The outputs of the LUTs are fed to the carry block. When the LUTs are in the normal mode of operation, the P outputs are directly connected to the corresponding carry block outputs. When the LUTs are in the arithmetic mode of operation, each of the arithmetic LUT behaves as a bit-slice of a carry look-ahead adder. The carry block generates carry signals based on

the P outputs and the G outputs of the arithmetic LUTs. In the arithmetic mode, each output of the carry block represents a bit of the sum output of a carry look-ahead adder. Each carry block output is connected to a flip-flop input. Each two-input multiplexers is used to select either a carry block output or the corresponding flip-flop output to produce a logic block output. Note that all four two-input multiplexers also share a single SRAM bit that stores their programmable configuration.

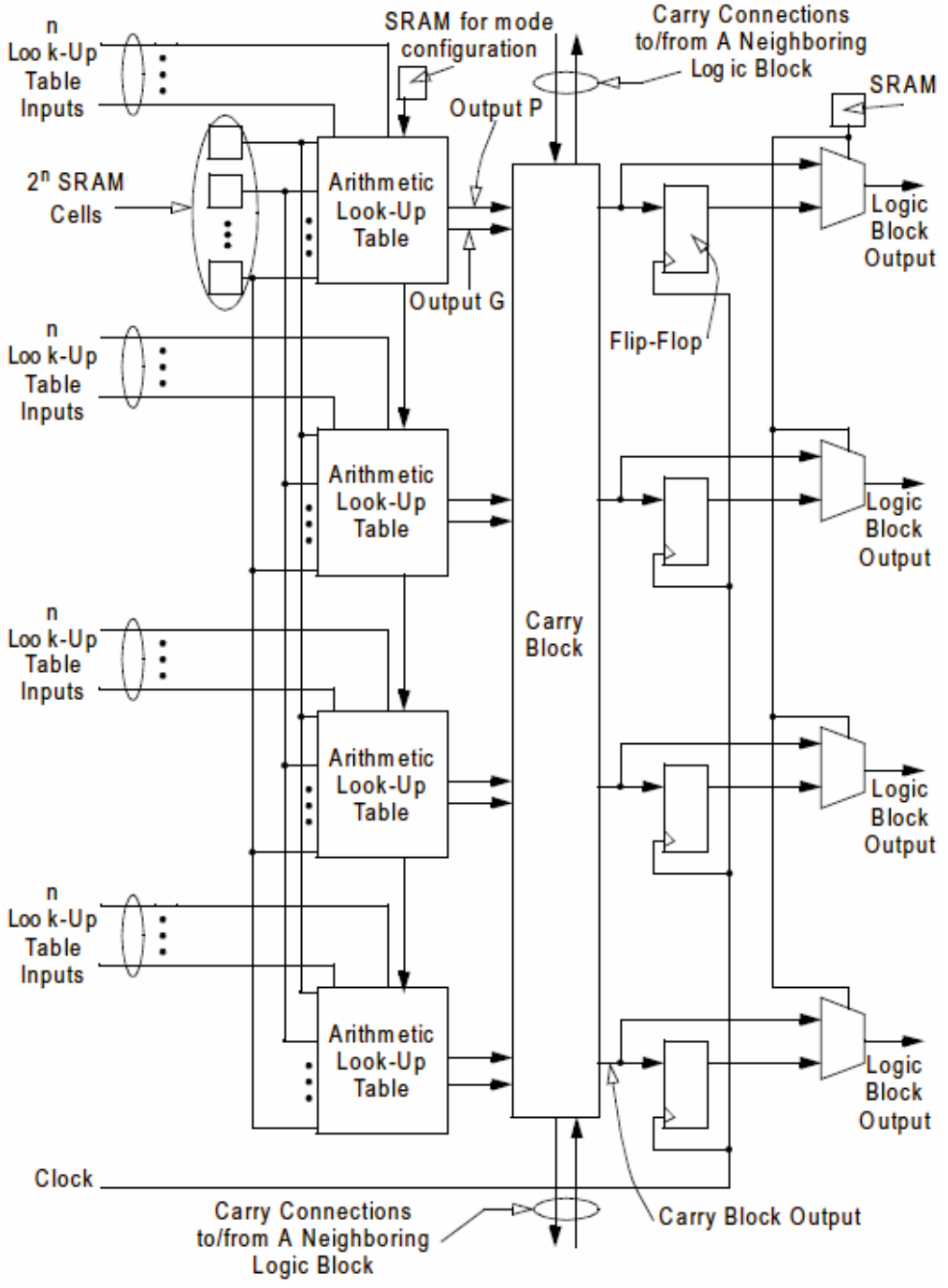

Figure 5: DP-FPGA Logic Block

The shift block, shown in figure 6, is a barrel shifter and is design to perform arithmetic and logical shift operations, which are commonly found in arithmetic applications, for multiple-bit wide

data. The block is necessary for the DP-FPGA architecture due to the limited connectivity provided by the data connection blocks shown above. Without the shift blocks, the DP-FPGA architecture is only capable of performing coarse-grain shift operations in increments of four. Such limitation can greatly reduce the usefulness of the architecture. In order to accommodate all possible shift operations, the shift block is elaborately designed. It can either left shift or right shift the output of the logic block and presents the shifted data at its output. The output is then connected to the data sub-channel through the output data connection block. For each shift operation, new data can be shifted in from several sources including the outputs of the logic blocks above and below, constant 0s, and constant 1s.

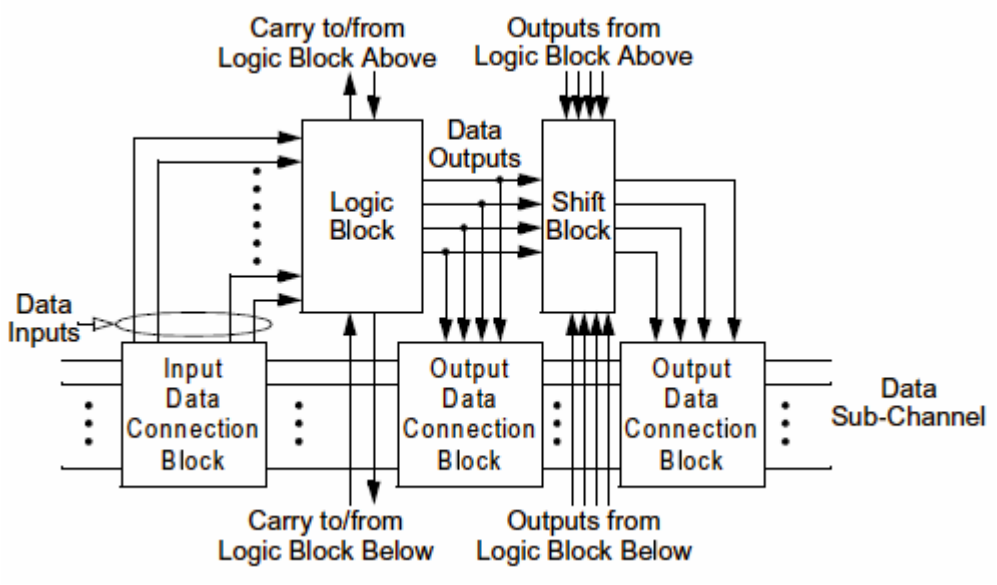

Figure 6: Logic block connectivity and the shift block of DP-FPGA

Several shift blocks can be used to shift multiple-bit wide data generated by multiple logic blocks when these logic blocks are physically placed adjacent to one another. The placement is necessary in order to allow data to be cascaded from one shift block to another through the outputs from logic block above and the outputs from logic block below connections shown in Figure 6.

4.4.2 COMPUTATIONAL FIELD PROGRAMMABLE ARCHITECTURE

The core block used in CFPA to implement data path operations is called a Partial Add, Subtract, and Multiply (PASM) block. The PASM block is formed in a nibble structure and operates on 4-bit operands. Each block implements a ripple carry addition, subtraction and partial multiplication. The PASMs can be connected together through the interconnection resources to implement adders, subtractors, and multipliers of any size. The PASM can also realize several bit-wise logical operations, including AND, XOR and NOR. Figure 7(a) shows a simplified schematic of the PASM block, consisting of 4 bit-slices. Each bit-slice has 2 data inputs and a single data output. Carry-in (cin) and multiplication (m) input bits are also shown, along with the carry-out (cout) bit. The core of a bit-slice is a full adder, represented by the 2 XOR gates and carry logic, shown in Figure 7(b). The AND gates and multiplexers permit the PASM to be configured as an adder, subtractor, or partial multiplier, as dictated by 9 SRAMs that are not shown in the figure. The comparator logic provides 3 outputs: greater than, less than, and equal. These outputs can also be programmed to generate the less than or equal, greater than or equal, and not equal functions. The

comparator outputs are multiplexed onto the out1, out2, and out3 pins of the PASM block. Output logic contains a multiplexer to select the output, an edge triggered D flip-flop with asynchronous reset, and output drivers. In total, there are 4 flip-flops in each PASM that can be combined to form a 4-bit register. This register can be shifted in both directions based on the programming of the SRAM bits within a PASM and its routing. The low number of SRAM bits simplifies the programming of the PASM block, which is also desirable for reconfigurable computing. Details of the PASM block can be found in [22].

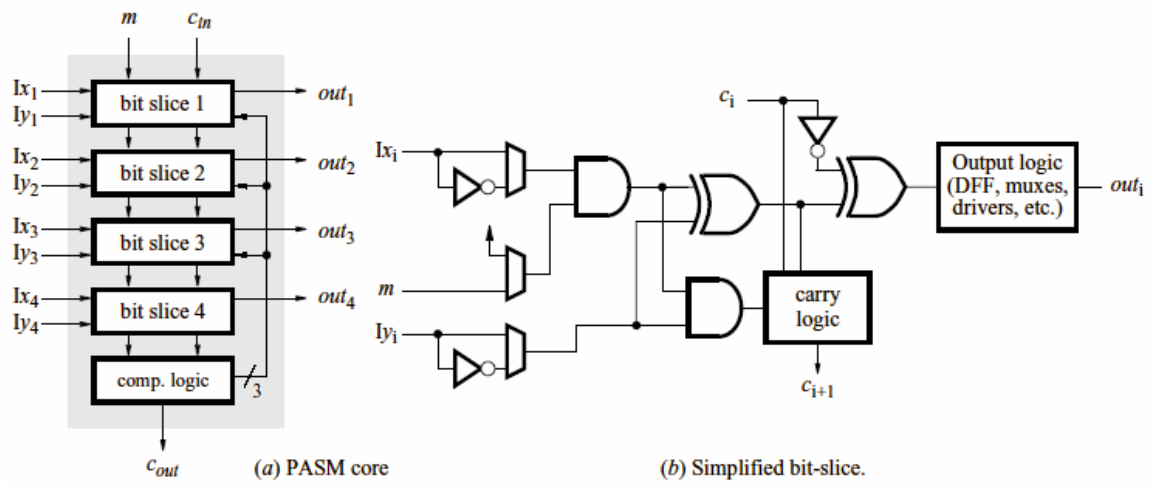

Figure 7: PASM core block of CFPA

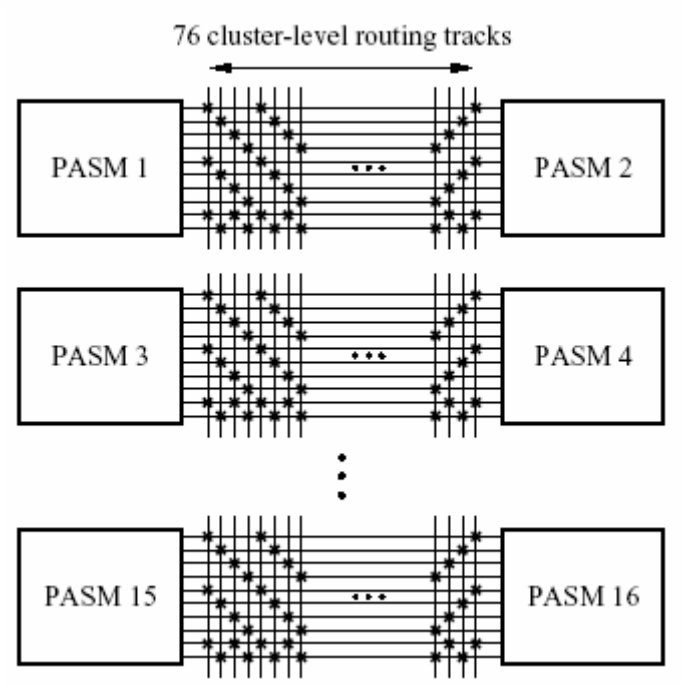

Figure 8: Routing resources in a CFPA cluster

The hierarchical routing architecture of CFPA is similar in structure to some commercial FPDs such as Altera's Flex family [32]. The lowest level of hierarchy in CFPA is called a cluster. Figure 8 shows a CFPA cluster that contains 16 PASMs. The inputs of each PASM are connected to cluster

level tracks through a partially populated crossbar. There are a total of 76 tracks in the cluster-level routing channel that can be connected to the 10 inputs of a PASM block using more than 130 switches. Each input of the PASM can be connected to 13 tracks of a cluster channel, on average. The switches that allow the connection of the tracks in a cluster channel to each input of the PASM are implemented using a multiplexer. This reduces the number of SRAMs that are needed to program the partially populated crossbar. Although the programming of the PASM block is nibble-controlled, the inputs of the block can be connected to the cluster channel independently. This provides higher routability with the cost of larger interconnection area. There are also dedicated routing resources that provide fast and area-efficient connections between adjacent PASMs. These dedicated resources, which are not shown in Figure 8, simplify the implementation of multipliers, fast adders, and shift registers.

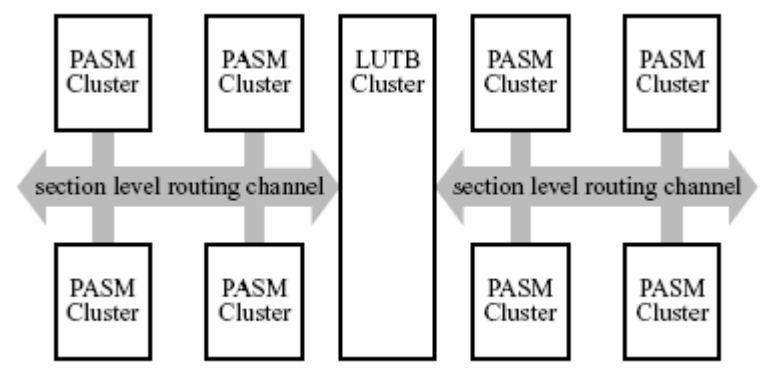

Figure 9: A section of the overall structure of a CFPA

To support control circuitry that cannot be efficiently mapped to PASM blocks, CFPA also contains LUT Blocks (LUTBs) that can implement the control part of a circuit. Each LUTB contains four 4-input LUTs that are connected through a fully-populated crossbar. 16 LUTBs are connected together in a cluster to provide a collection of 64 LUTs in total. The LUTB cluster is similar to commercial FPGAs in many aspects. Figure 9 shows a section of CFPA that contains a LUTB cluster and several PASM clusters. LUTs can connect to PASMs through 2 levels of hierarchy: section-level routing channels and cluster-level routing channels.

4.4.3 MULTI-BIT FPGA (MB-FPGA)

MB-FPGA is a recent attempt to define an architecture for a datapath oriented FPGA along with a complete set of tools to map applications on the architecture. It builds upon the deficiencies of the DP-FPGA. It uses the concept of super-clusters, each of which contains clusters of a basic logic element (BLE). Each BLE is basically a 4-LUT based structure with enhanced carry-chaining support. The details of MB-FPGA can be found in [9].

4.5 DATAPATH-ORIENTED FEATURES ON COMMERCIAL FPGAS

Commercially available general-purpose FPGAs have been incrementally adding datapath oriented features throughout the years. The earliest adopted datapath oriented features is carry chains; and in recent years, more complex datapath-oriented features like DSP blocks [8] (Altera Stratix and Stratix II) and multipliers [7] (Xilinx Virtex II and Virtex II Pro) have been added to existing architectures. Unlike the application domain tuning done in this work, however, these features are mainly aimed at improving the performance of specific arithmetic functions through heterogeneous architectures. The heterogeneity introduced by the DSP and multiplier blocks often

increases the complexity of FPGA CAD flow, which potentially can introduce inefficiency in area utilization. Finally, even though routing area accounts for a majority of the total FPGA area, none of the existing commercial FPGAs utilizes the CMS routing resources, which employs configuration memory sharing to increase the area efficiency of routing resources in datapath oriented applications. Furthermore, little research has been done on designing automated CAD tools that can capture datapath regularity and efficiently utilize the regularity on CMS logic or routing resources.

5. DSP OPTIMIZED LOGIC BLOCK ARCHITECTURES

5.1 CLASSIFICATION OF OPTIMIZATION TECHNIQUES

Based on the chosen optimization technique, a slightly different but closely related classification of the architectures can be identified. This classification applies to the dynamic ALU-based and the LUT-based architectures only.

1. **Architectures with dedicated DSP logic:** The DSP-tuning is achieved by the use of hardwired logic which implements datapath functions. Dependent on the amount of the dedicated logic resources and their overall organization, FPGA architectures of this kind have either a homogeneous structure (all logic blocks are DSP-optimized) [23] or are heterogeneous (an FPGA is divided into regions with DSP-optimized and general-purpose cells) [21]. The third group includes hybrid FPGA architectures which are globally homogeneous and locally heterogeneous (an FPGA has either a hierarchical structure with a heterogeneous lowest level [22], or each FPGA cell has mixed-type components [24]).

2. **Architectures of coarse granularity:** These architectures operate on wider (multi-bit) arguments. Because of the cost-efficiency tradeoff it offers [25], a 4-bit processing has been particularly popular [5] [21] [26]. Also, in many commercial general-purpose FPGA devices very coarse logic blocks have been used to allow the generation of multi-bit results in one processing element, and, at the same time, the reduction of the routing resource complexity [7] [8].

3. **Architectures with DSP-specific improvements:** In some FPGA architectures, only small adjustments to the conventional structure have been made to efficiently support DSP functionality. One of the most popular adjustments of this type is the use of dedicated carry logic [35]. The carry logic is faster than the carry signal generated in a LUT. Another technique is sharing of the logic block inputs which allows a multi-bit output to be produced from one set of inputs only (common in datapaths). This technique has been applied both in traditional LUTs [34] and in multi-bit output LUTs [26] [27]. Another adjustment technique exploits the fact that slices of datapaths usually implement the same functionality. Thus, configuration bit sharing can be applied to reduce unnecessary area overhead. This technique has been proposed in [21].

5.2 SOME REPRESENTATIVE LOGIC BLOCK ARCHITECTURES

5.2.1 THE MIXED-GRAIN LOGIC BLOCK

The mixed-grain architecture as proposed in [1] contains a single 4-LUT in each of its logic blocks. The logic blocks can be configured into two modes including the random logic mode and the arithmetic mode. In the random-logic mode, the logic block behaves as a single 4-LUT. In the arithmetic mode, the 4-LUT is decomposed into 4x2-LUTs in order to implement four distinct bit-slices. The full adder is realized using just 4 bits of LUT by exploiting the inverse symmetry property

of the full-adder truth table as shown in figure 10. The basic structure of each full adder slice is shown in figure 11.

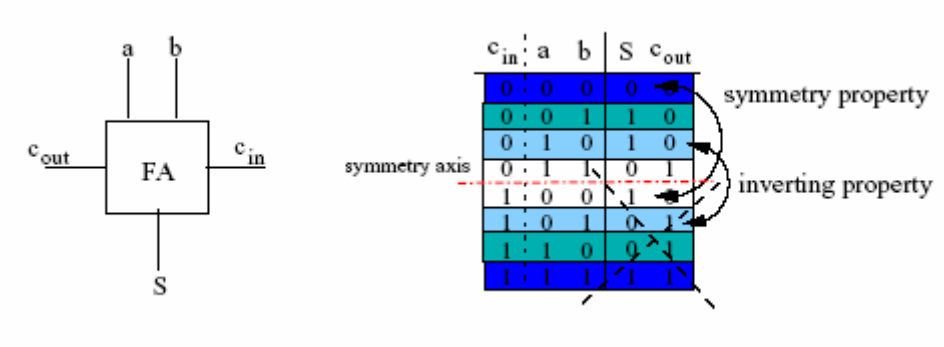

Figure 10: Inverting and symmetry properties of a binary adder identified in its truth table.

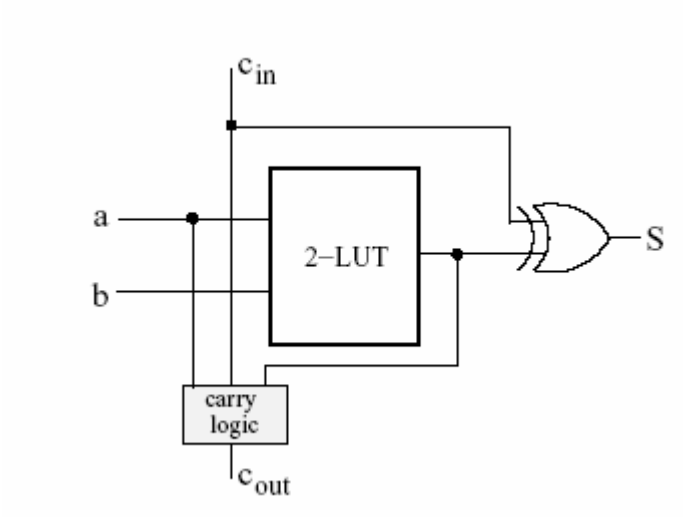

Figure 11: An optimal 1-bit LUT-based adder implementation

Bit-Slice Structure

A basic logic element of the proposed logic block implements a bit-slice of datapath functions. The logic element is based on the structure shown in figure 11, and consists of a 2-LUT, an XOR gate and a dedicated carry logic circuitry. To allow an implementation of arithmetic functions other than a 1-bit addition for which this structure has been optimized, two extra gates, i.e. an XOR gate and AND gate, are placed at the LUT inputs. The AND gate is used in a multiplier-mode and allows an array multiplier cell to be implemented in a single bit-slice of the logic block. The gate implements a logical AND of the data input Ai and the global signal Y. The XOR gate at the second LUT input is used for the implementation of the addition or subtraction operations, which can be chosen dynamically by setting the global signal Z (Z = ADD=SUB). The signal Z and the data input B_i are inputs of the XOR gate.

The results of the application-domain characterization discussed in Section 3 reveal the presence of a considerable number of multiplexers in DSP designs. The importance of having support for the efficient implementation of multiplexers in an FPGA logic block has also been confirmed by other studies [23] [21]. Therefore, a bit-slice also allows mapping of a 2:1 multiplexer. However, the multiplexer function ($F = a \bullet \bar{c} + b \bullet c$, where 'a' and 'b' are multiplexer inputs and 'c' is a selection signal) is not symmetrical, and its mapping onto the proposed logic element is not directly possible. Since placing of a dedicated 2:1 multiplexer in the proposed logic element is better from both the area as well as performance point of view, this option is chosen. The 2:1 logic multiplexer (LMUX in figure 12) is placed in parallel to the 2-LUT. Its inputs are two data inputs of the logic element (i.e. A_i and B_i), and a selection signal is the global signal X. The structure of the logic element is shown in figure 12. The element has two outputs: a datapath output D_i and a random logic output L_i. The D_i output is used when the logic element is configured as a datapath bit-slice and a multi-bit result is generated, while the L_i output is used in the random logic mode when a single-bit result is produced. The LMUX, 2-LUT and XOR gate outputs are multiplexed to allow an implementation of wide Boolean functions or wide multiplexers (both generate multi-bit output).

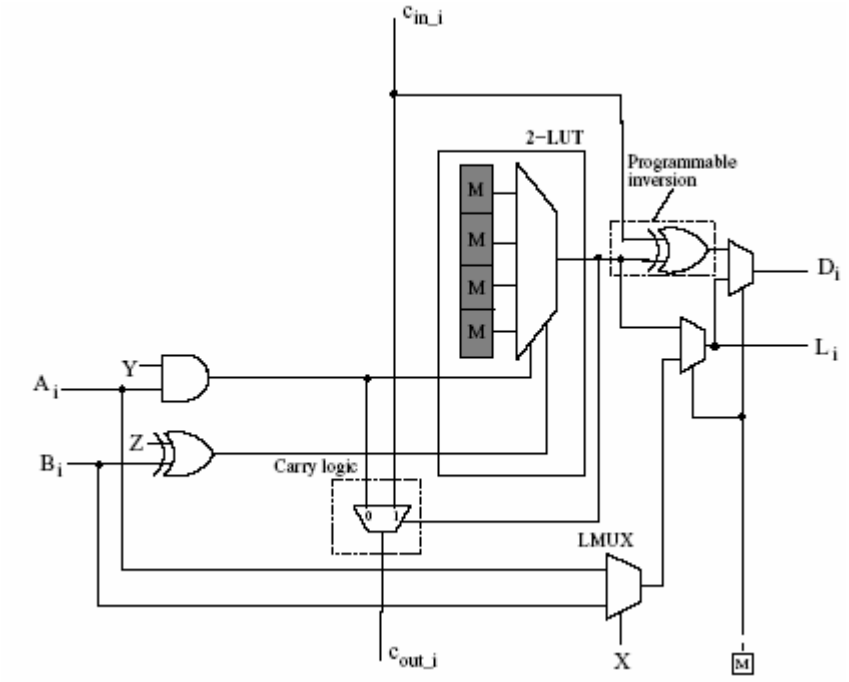

Figure 12: Basic logic element of the mixed-grain logic block, which implements a bit slice of datapath functions.

Logic Block Structure

The mixed-grain logic block consists of four logic elements (slices), with a structure as shown in figure 12. The choice of 4-bits as a granularity of the logic block is dictated by the fact that such granularity has been found the most optimal for the datapath mapping [21] [25]. The logic outputs of each slice are merged together in global multiplexers MUX1, MUX2 and MUX3, while datapath outputs are fed to the output selection stage (figure 13). Additionally, the outputs of multiplexers MUX1 and MUX2 are fed to the output selection stage, where they are multiplexed with datapath outputs of the first and third bit-slice, respectively. These outputs are used if a 2-bit 4:1 multiplexer is implemented.

The logic block has eight primary inputs IN_0 - IN_7, three control inputs C_0, C_1, C_2, four primary outputs OUT_0-OUT_3, and a special carry output, COUT. The control signals are multiplexed with the static signals '0' and '1' defined by the configuration bits, and produce a set of signals X, Y, Z, respectively. The X, Y, Z signals are global for all bit-slice. The sharing of these signals is possible because of the datapath regularities. As a result, this allows reducing the number of logic block pins, and consequently the routing resource complexity. The carry input signal CIN on pin C_0 is fed to the first bit slice only. This slice produces an intermediate carry output signal cout0 which is connected with the carry input of the next (second) bit-slice, and so on. The task of the input selection stage is to define the connectivity between inputs of the logic block and inputs of each logic element (LUT) dependent on the required functional mode. A similar function the output selection stage has. Each of the logic block outputs can be registered by means of dedicated flip-flops.

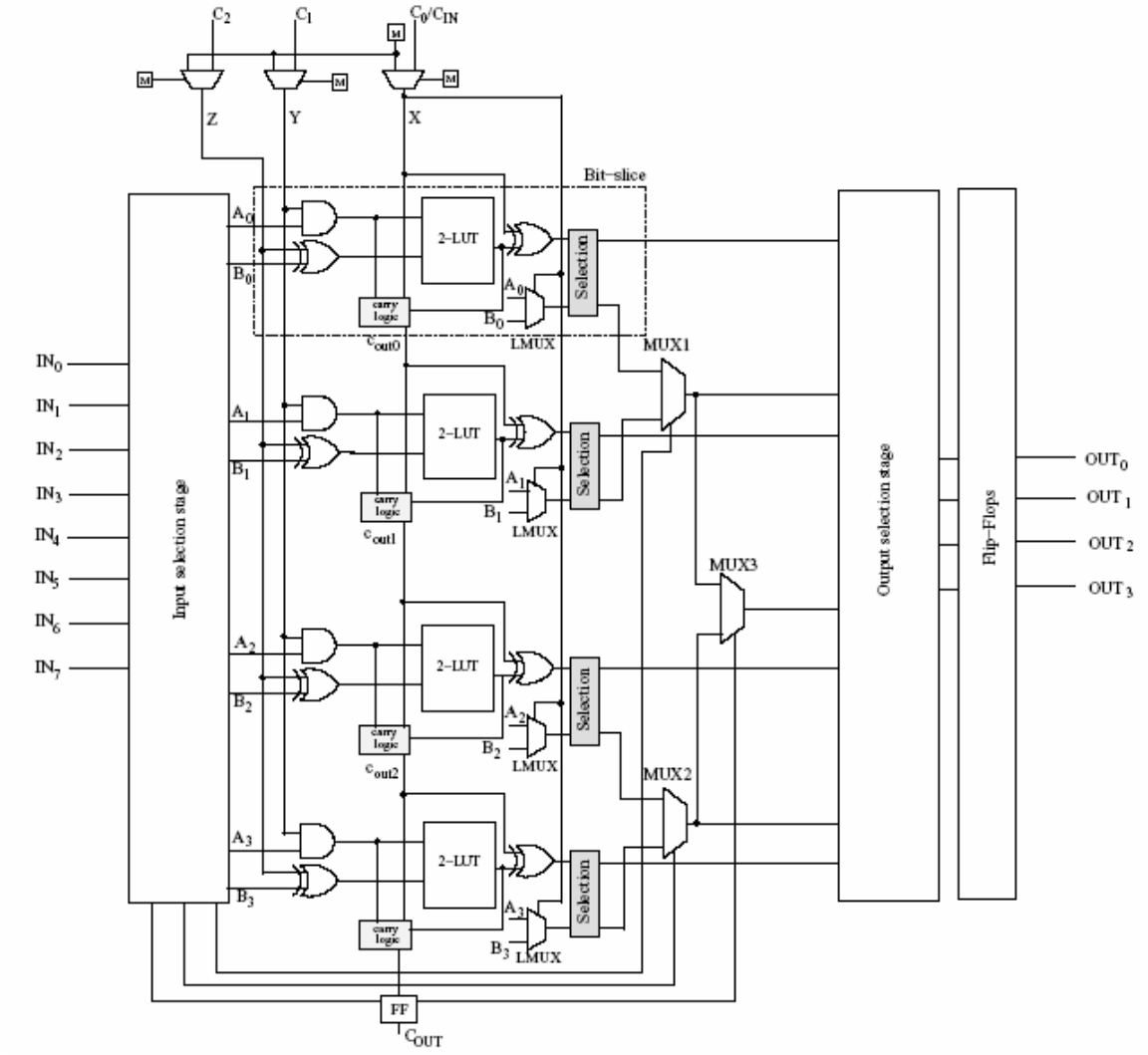

Figure 13: The complete mixed-grain logic block architecture

Functional Modes

The mixed-grain logic block has two primary functional modes:

1. Data-path mode in which a multi-bit output (max. 4-bit) is produced using D_i outputs of each logic element; all bit-slices are configured to implement the same function.

2. Random logic mode in which a single-bit output is produced using L_i outputs of each logic element; bit-slices usually implement different logic functions (see figure 14).

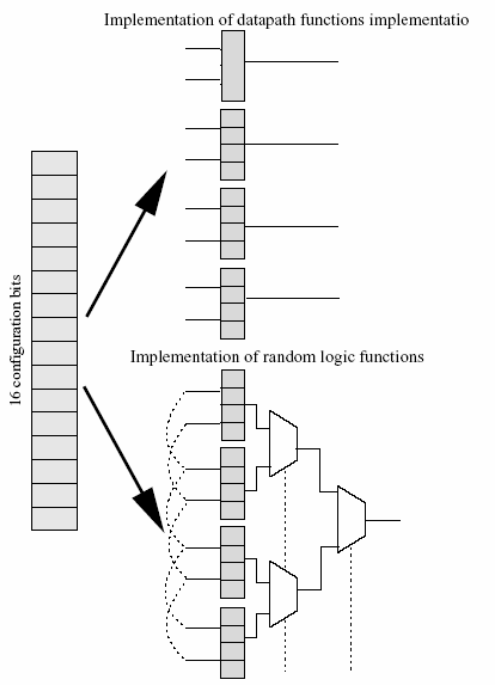

Figure 14: Reuse of the LUT configuration bits to support implementation of two different modes of computations.

It is assumed that only one of these modes is possible at the same time. If the number of bit-slices required to implement a given function is smaller than the total number of the logic block slices (i.e. four), the remaining logic block resources are left unused. Although this might slightly influence the total number of logic blocks required to implement such functionality, it reduces the complexity of the control structure in each logic block. Below, the logic block functionality in each of the modes is discussed in detail. Figure 15 shows connectivity examples in each mode.

1. Datapath Mode

Addition/Subtraction: The logic block can be configured to implement up to 4-bit addition/subtraction operations. The type of an operation can be determined statically or dynamically by connecting the signal Z = ADD/SUB to a local or global signal, respectively. A 2-LUT in each bit-slice stores half a truth table of a 1-bit adder. If a binary addition is implemented, the input XOR gate passes the input operand Bi without changing its polarization, i.e. Z='0'. If a subtraction is implemented, Z='1' and the XOR gate negates one of the operands. The input AND gates are unused, i.e. signal Y='1'. The dedicated carry logic implements the carry path. For operations with less than four bits, unused slices have their LUTs programmed to generate the value '1' such that the carry logic allows the carry signal from the previous slices to be propagated to the output (COUT). The outputs (sum/decrement bits) are available on the datapath outputs of each logic element.

Multiplication: The logic block supports an implementation of an array multiplier with a ripple carry adder [28]. Maximally, a 4-bit section of such a multiplier (i.e. four cells) can be implemented in one logic block. For this purpose, each bit-slice is configured as a binary adder (inactive XOR gates, i.e. Z='0') with an AND gate on one of its inputs. The signal Y carries the value of the multiplicand bit. In this mode, four partial product signals are generated on the datapath outputs, and the carry output signal is available on the COUT pin.

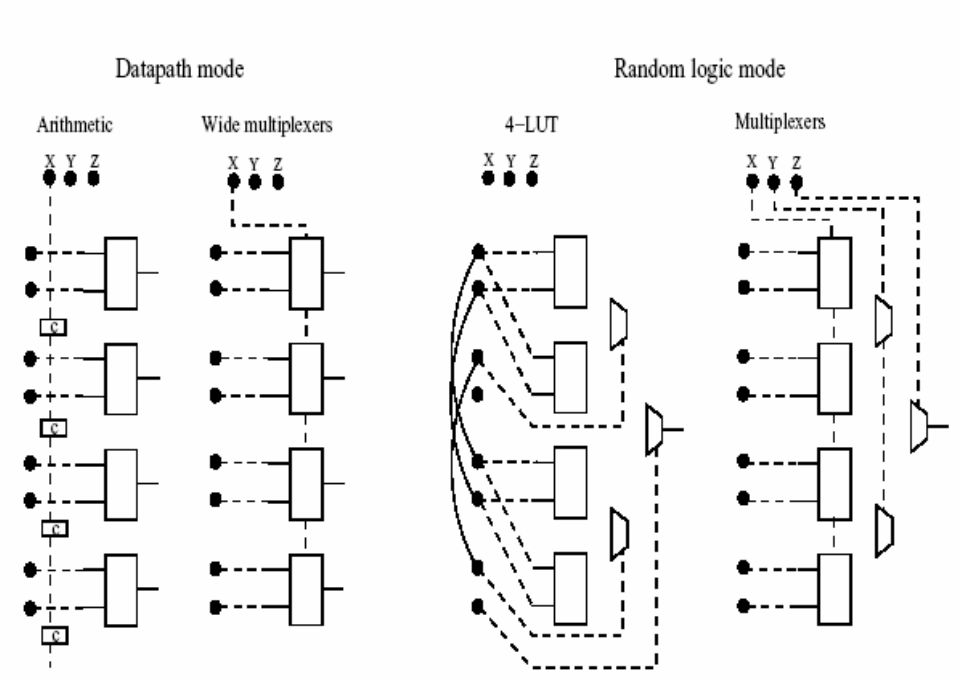

Figure 15: The connectivity between inputs of the logic block and inputs of the consecutive bit-slices.

Wide multiplexers: In the datapath mode, the multiplexers in the logic block have multi-bit inputs and outputs. Maximally, a 4- bit 2:1 multiplexer or a 2-bit 4:1 multiplexer can be implemented in one logic block. In the first case, LMUXes of each bit-slice are used, and the result is available on the datapath outputs. In the second case, each slice produces output bits on the logic outputs of each bit slice, and the global multiplexers MUX1 and MUX2 are used to generate two final signals. The result is available on the first and third output pins of the logic block. A wide 2:1 multiplexer uses global signal X as a control signal, and a wide 4:1 multiplexer uses signals X and Y as control signals.

Wide Boolean functions: The logic block allows the implementation of Boolean functions of two multi-bit operands. Such functions are implemented by storing the same truth table of a given Boolean function in each 2-LUT. In each bit-slice, the LUT output is selected in an upper selection multiplexer, and connected to the datapath output.

2. Random Logic Mode

Boolean functions: Logic (Boolean) functions of up to four inputs can be implemented in the logic block. A 4-input function is decomposed using Shannon expansion theory [29], and mapped onto a set of four 2-LUTs and global multiplexers MUX1, MUX2 and MUX3, which

merge the LUT outputs. Each bit-slice produces a result available on the logic output. The 3-input/2-output logic functions can also be realized in a single logic block.

Multiplexers: The mapping of multiplexers with a single output and up to eight inputs is possible. If the largest multiplexer, i.e. a 1-bit 8:1 MUX, is to be implemented in the logic block, all global multiplexers MUX1, MUX2 and MUX3 are used. In this case, LMUXes in each slice and multiplexers MUX1-MUX2 and MUX3 are controlled by global X;Y, and Z signals, respectively. The data inputs of the multiplexers are inputs of the logic block, and the final output is available on all logic block outputs.

Interconnect Structure

The global routing network of the mixed-grain architecture is similar to the conventional FPGA architecture in topology. The routing tracks, however, are grouped into 2-bit wide buses. In the random-logic mode, the bus is used to route a single bit of data; and in the arithmetic mode, the bus is used to route two bits of data at a time. The architecture also contains some special routing tracks, each independently controlled by a single set of configuration memory, dedicated for routing the control signals of the logic blocks (Figure 16).

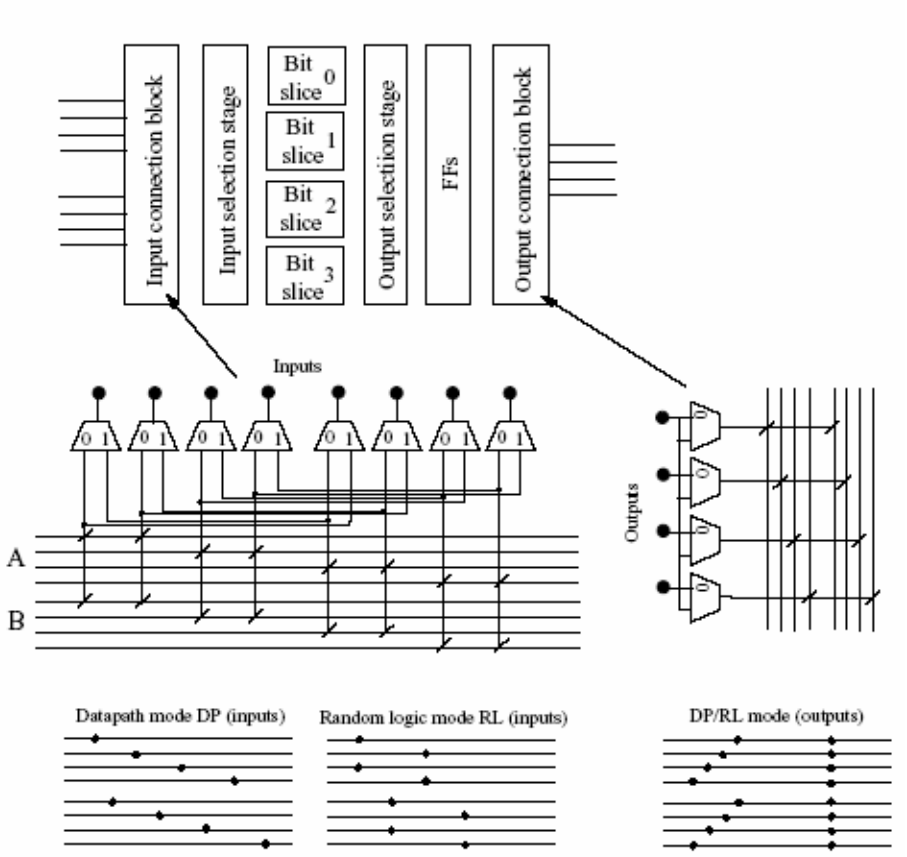

Figure 16: The proposed implementation of the programmable interconnects.

Overall Assessments

Although the work in [1] has defined the basics of the mix-grain FPGA routing architecture, the work did not measure its area efficiency by actually placing and routing a set of benchmark circuits

on the architecture. Instead, the routing area required to implement several simple benchmarks are roughly estimated by counting the total number of logic block pins required to implement each circuit. This method is much less accurate than area measurements obtained by actually implementing a set of well-chosen benchmarks on a given architecture [Betz99a]. Furthermore, the feasibility of the routing architecture has not been exactly proven since no circuit has been actually implementing on the architecture.

5.2.2 BIT-SERIAL LOGIC BLOCK WITH LUT EMBEDDING A SHIFT REGISTER FUNCTIONALITY

Bit-serial DSP architectures are attractive for conventional FPGAs because of the fact that there is no need to increase the routing resources while increasing the gate capacity for implementing bit-serial datapaths [3]. The most significant reason for this is the high routability of bit-serial circuits. This is best described by referring to the Rent's Rule. The Rent exponents of various bit-serial circuits are observed to be between 0.22 and 0.37. It is known that if the Rent exponent is below 0.5, the expected average wiring length becomes independent on the circuit size, whereas if the Rent exponent is higher than 0.5, the average wiring length increases as the circuit size grows.

Based on these observations, a bit-serial FPGA logic block architecture is proposed which incorporates a LUT structure that efficiently implements shift registers so that the large amounts of shift registers that are needed to synchronize the data streams in a bit-serial design can be implemented using the LUT instead of the D-flip flops.

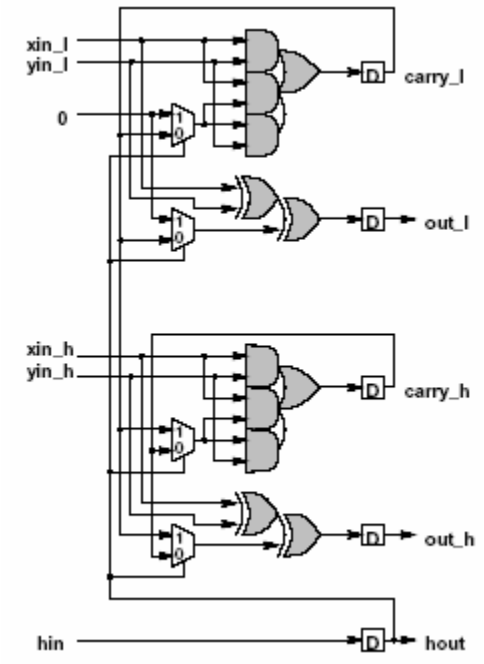

Figure 17: A double-precision bit-serial adder implementation.

Bit-Serial Building Blocks

Bit-serial operations process the input one bit at a time. Therefore bit-serial architecture simplifies the hardware since the entire bits pass through a single bit wide module. Moreover, bit-serial architecture can operate at a higher clock frequency than bit-parallel architecture since it takes

no account of carry propagation chain. In the case of FPGAs, signal routing delay is significant factor and it determines the system delay performance. Bit-serial architecture tends to have very localized routing, often to only a few local destinations. In contrast, the bit-parallel architecture requires many modules, so the routing resources often are insufficient (low utilization) or the resulting design is too slow (large routing delays). The bit-serial architecture fits more efficiently in a FPGA which has limited routing resource.

In add operation, two input data operands are shifted from LSB at the same time. The carry is stored in register and then fed back to the carry input of full adder. Unlike bit-parallel operation, carry propagation does not occur (carry is saved). The output of the circuit is stored in registers for bit pipelining. The latency is one bit time. In the proposed bit-serial FPGA, this adder is implemented by half CLB and a double precision bit-serial adder is implemented by one CLB. A double-precision bit-serial adder is shown in figure 17.

Bit-serial multiplier can produce a 2N-bit double precision product in every N clock cycles for N-bit inputs. Bit-serial double precision data is represented by two wires. In both cases, the output is bit-serial, with the LSB first. This structure is suited for FPGAs because of its routing to nearest neighbors with inputs and outputs. The number of cells is proportional to the input data width. A bit-serial multiplier building block is shown in figure 18.

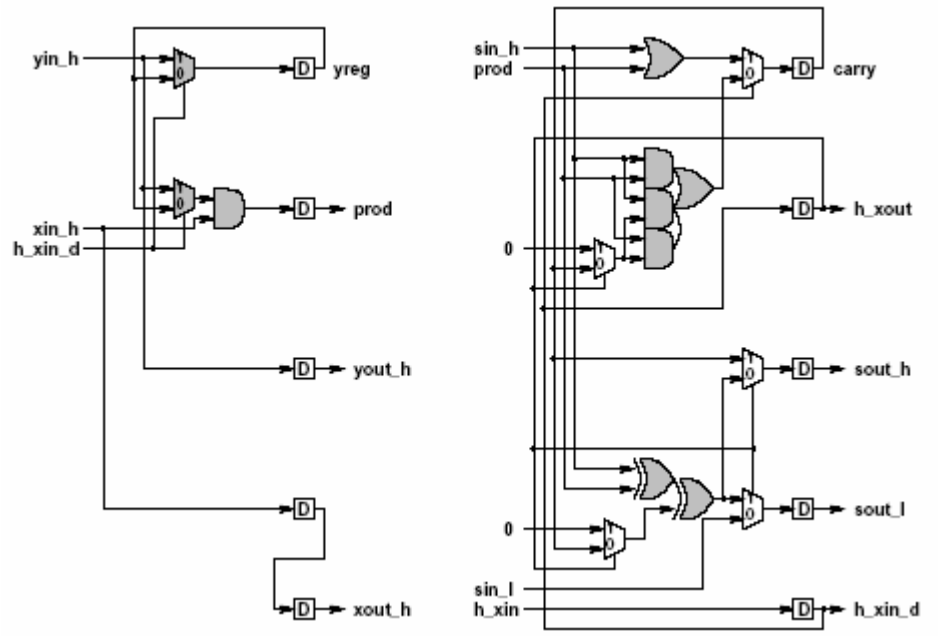

Figure 18: A bit-serial multiplier building block.

The Bit-Serial Logic Block

One of the distinctive characteristics of bit-serial circuits is that the connectivity inside the cell is dense, while the connectivity between bit-serial cells is sparse. Therefore, the strategy used is to increase the logic capacity of the logic block and absorb the dense interconnection inside the logic block to reduce the inter-block routing resource. Figure 19 shows the logic block architecture. The following description summarizes the features of the logic block architecture:

- 4x4-input LUTs and 6 flip-flops.

- The two multiplexers in front of the LUTs are targeted mainly for carry-save operations which are frequently used in bit-serial computations.

- There are 18 signal inputs and 6 signal outputs, plus a clock input.

- Feed-back inputs c2, c3, c4, c5 can be connected to either GND or VDD or to one of the 4 outputs d0, d1, d2, d3. Therefore, each LUT can implement any 4-input functions controlled by inputs a0, a1, a2, a3 or b0, b1, b2, b3.

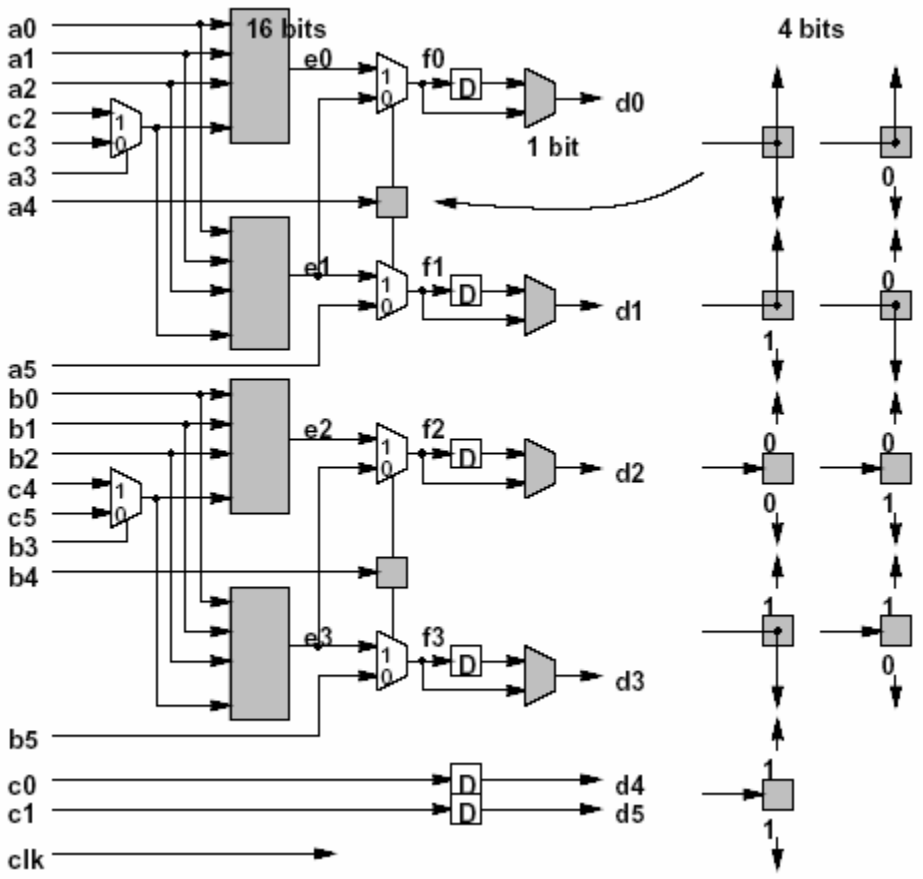

Figure 19: The bit-serial logic block architecture.

- Programmable switches connected to inputs a4 and b4 control the functionality of the four multiplexers at the output of LUTs. As a result, 2 LUTs can implement any 5-input functions.

- The final outputs d0, d1, d2, d3 can either be the direct outputs from the multiplexers or the outputs from flip-flops. All bit-serial operators use the outputs from flip-flops; therefore the attached programmable switches are actually unnecessary. They are only present in order to implement any other logic functions other than bit-serial datapath circuits.

- Two flip-flops are added (inputs c0 and c1) to implement shift registers which are frequently used in bit-serial operations.

LUT Architecture with Shift Register Function

Bit-serial datapath requires many shift registers for pipeline synchronization because each pipeline path may have different latency. For example, an adder's latency is 1 while N-bit double precision multiplier latency is 2N - 1. The original logic block has two extra flip-flops in each CLB. In one CLB, 6 clocks shift can be implemented. But this is still not enough for many of the bit-serial applications such as DCT, adaptive filters, recursive filters, etc. The solution is a new lookup table architecture which can be configured as a shift register or as combinatorial logic.

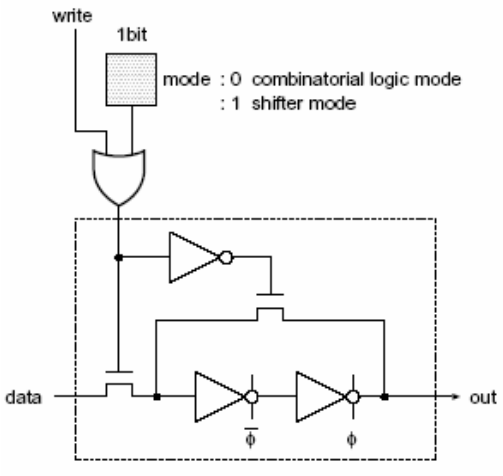

Figure 20: The LUT RAM cell.

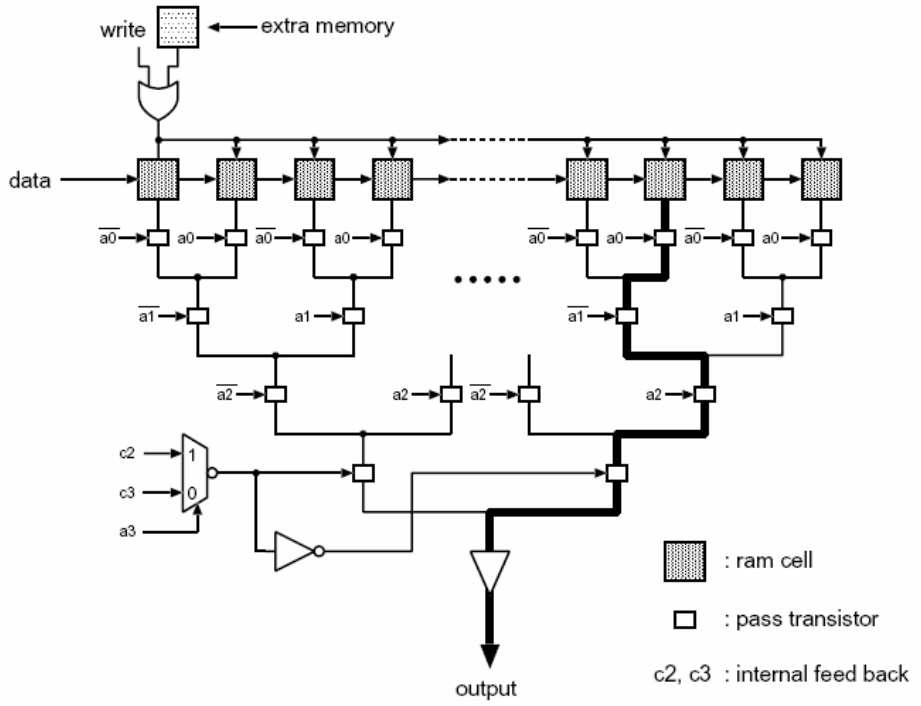

Figure 21: Bit-serial 4-LUT implementing shift register function

Figure 20 shows the LUT RAM cell. The LUT is constructed from a series of these RAM cells and a decoder. An LUT has an extra 1-bit memory which configures the LUT as either

combinatorial logic or shift register mode. Data is stored in clocked inverter loop when write signal is high. The control signal write is generated by the configuration logic and is shared by four LUTs within one CLB. Thus, with this RAM cell structure, either a chain of shifters or a look-up table can be implemented.

Figure 21 shows a 4-input LUT using the modified RAM cell. The modified LUT is similar to the original LUT except for the RAM cell. a0, a1, a2 and a3 are input signals from interconnect network and c2, c3 are internal feedback signals. A 4-input LUT requires 16 bit memory and a 16 to 1 decoder which has a binary tree structure.

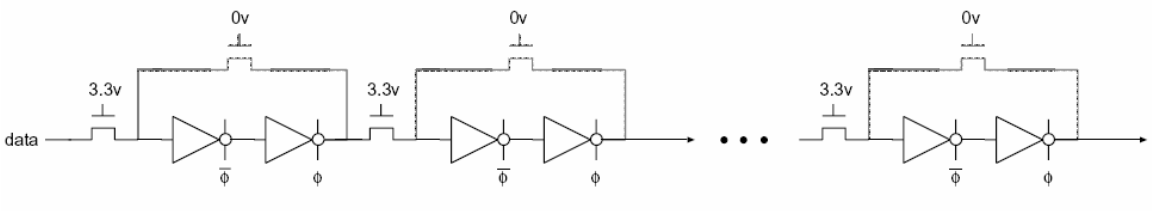

Figure 22: Configuration and shifter mode.

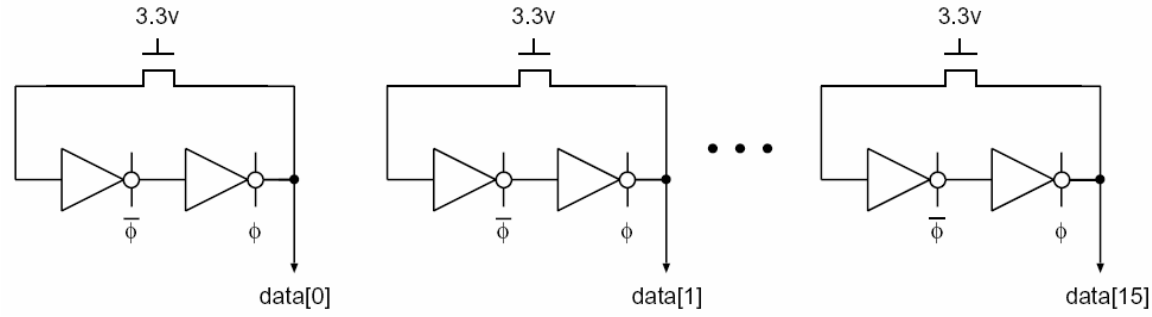

Figure 23: Combinatorial logic mode.

Figure 22 shows the RAM cells in configuration and shifter mode. When the RAM cells are used as a lookup table, the configuration data are shifted in a bit-serial manner from the data input. After all the configuration data for the LUT is transmitted, the shift register chain is disconnected and configuration data are stored at each RAM cell (figure 23). When the RAM cells are used as shift registers, shift register chain is preserved during operation mode. LUT inputs a0, a1, a2, a3 and c2, c3 determine the number of shifts. For example, when a0 = a2 = a3 = c1 = 1 and a1 = c3 = 0, LUT behaves as a 14 bit shifter (bold line in figure 21). In shift register mode, a0, a1, a2 and a3 are fixed to '0' or '1'.

Routing Architecture

The overall routing architecture resembles that of a Xilinx 4000 series FPGAs. Further, a two-level routing structure has been proposed to take care of the dense internal routing in a logic block and the reduced external routing requirements of bit-serial architectures. The features of this routing architecture are as follows:

1. The large number of input and output signals of the logic block would create a significant capacitive load due to the drain capacitance of the pass-transistors on the routing segments. This routing scheme is seen in Xilinx 4000 FPGAs. In that scheme, the input and output

pins are connected directly onto the routing segments via pass-transistors. By using the routing scheme proposed, the intermediate routing resource inside the logic block (hence the two-level routing) enables insertion of buffers at the logic block pins which effectively isolates the capacitive load of the drain capacitance of pass-transistors from the routing segments. The routing delay through the single-length routing segment is greatly reduced. This fact also leads to power reduction.

2. All logic block outputs can be routed back to itself without consuming any external routing resource. Also, connections between adjacent logic blocks which frequently occurs in bit-serial circuits is implemented via connection-blocks without consuming any external routing resource as well.

3. The connectivity of inputs and outputs are made symmetric with a high degree of flexibility in four directions. This increases the routability and also makes the automatic routing very easy to develop.

Overall Evaluation

Although bit-serial designs are more suitable for many current commercial FPGA architectures in terms of its advantages such as efficient logic utilization and higher routability but if an FPGA architecture is to be re-designed from scratch, then digit-serial computation is a better choice for basic granularity of the computing elements. This is because a digit-serial architecture offers the advantages of area-efficiency of bit-serial architecture and the time-efficiency of bit-parallel architectures. Thus, a logic block which has been designed to directly map digit-serial blocks can take advantage of digit-serial structures.

5.2.3 THE DIGIT SERIAL LOGIC BLOCK ARCHITECTURE

In a digit-serial arithmetic implementation, the W bits of a data word are processed in units of the digit-size N in W/N clock cycles. This leads to arithmetic operators that have smaller area than equivalent parallel arithmetic versions and have a larger throughput than equivalent bit-serial arithmetic designs [4]. Architectures based on the digit-serial approach may offer the best overall solution for considering the tradeoffs between speed, efficient area utilization, throughput, I/O pin limitations and power consumption. Digit-serial approaches are best suited for implementation of digital signal processing systems which require moderate sampling rates.

Digit-Serial Building Blocks

A basic element in a digit-serial arithmetic implementation is the digit-serial adder (figure 24(a)) which adds two digits along with a previous carry bit and produces the sum digit and a new carry bit. A digit-level pipelined digit-serial multiplier with a fixed coefficient can be implemented by unfolding the structure of a bit-serial multiplier. One input of this multiplier is parallel while the other is digit-serial with the least significant digit presented first. The output is also digit-serial with the least significant digit first. The number of stages is proportional to the number of bits in the parallel input operand. An unsigned digit-level pipelined digit-serial multiplier module and multiplier structure with N=4 are shown in figure 24(b) and 24(c) respectively.

A two's complement digit-serial multiplier requires control circuitry to handle the sign extension of the multiplicand as well as the partial-products. A two's complement digit-serial multiplier with N=4 consists of the first, the middle and the last digit-serial multiplier modules, as shown in figure 25. The XOR gate in the last digit-serial multiplier module generates the sign extension. The last digit-serial multiplier module is used to add a one to the inverted LSB of the multiplicand X.

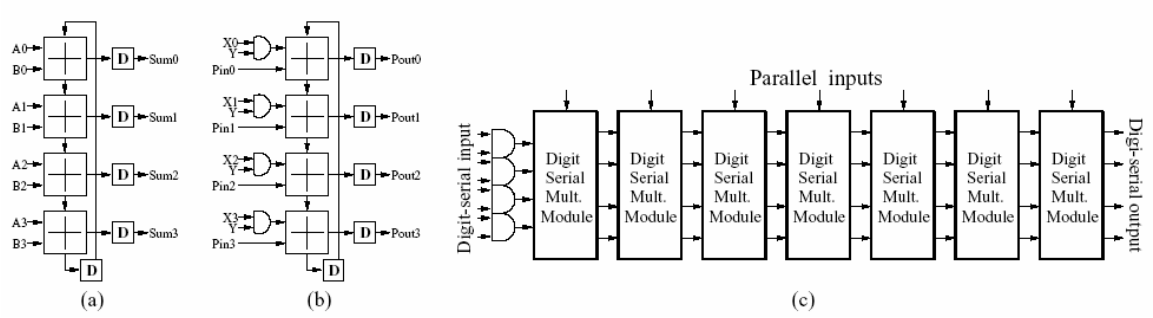

Figure 24: (a) Digit-serial adder (b) Digit-serial multiplier module (c) Un-signed digit-level pipelined digit-serial multiplier with word-length 8-bits and N=4.

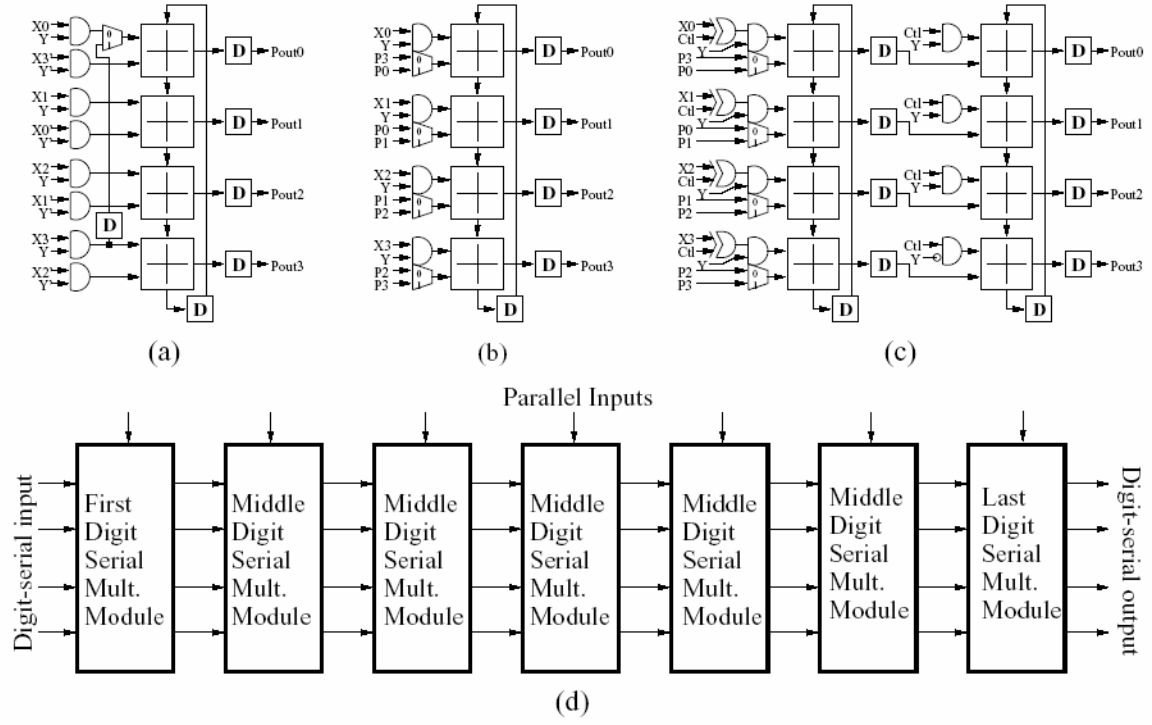

Figure 25: (a) First digit-serial multiplier module. (b) Middle digit-serial multiplier module. (c) Last digit-serial multiplier module. (d) Two's complement digit-level pipelined digit-serial multiplier with word-length 8-bits and N=4.

Digit-Serial Logic Block

The digit-serial logic block (DLB) architecture is a logic-based core cell which is simple and straightforward. Furthermore, a one-to-one correspondence between the logic design and the physical realization of digit-serial logic elements exists. This leads to direct implementation of basic mathematical functions in hardware, such as adders, subtractors and multipliers. The digit-serial logic block (DLB) architecture is composed of four main parts: digit-serial logic array, fast-carry logic carry-type select logic, and register array, as shown in figure 26. It has 26 inputs and 9 outputs. Static RAM programming technology is used in the digit-serial FPGA (DS-FPGA) to provide the in-circuit re-programmability. It will support efficient implementation of N=2, 3 and 4 digit-serial DSP

applications as well as random control circuits. The DLB architecture uniquely combines both fine and coarse logic granularity for optimum logic utilization and high performance. High logic utilization is provided by the fine-grained logic modules that can implement random logic functions without wasting device resources. The coarse-grained structure of the four fully interconnected logic modules provides fast operation and efficient routing with minimal signal skew for digit-serial DSP architectures.

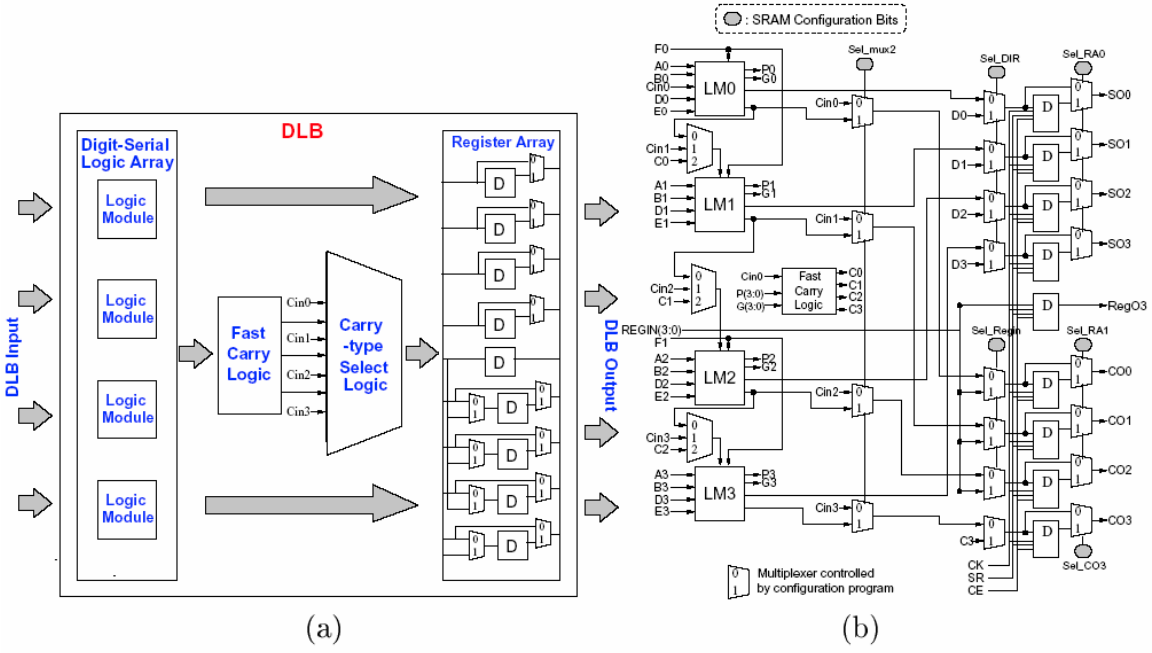

Figure 26: (a) Digit-serial logic block (DLB) diagram. (b) Details of digit-serial logic block architecture.

Digit-Serial Logic Array

The digit-serial logic array is composed of four small logic modules (LMs) which are the smallest unit of logic in the DLB architecture. Figure 27 depicts the structure for this logic module. A large number of logic functions can be implemented by using an appropriate subset of the inputs and tying the remaining inputs of a logic module high or low as shown in Table 1. Each logic module of DLB can be used independently for random logic implementation. A single logic array can implement either an N=2, 3 and 4 digit-serial adder/subtractor, unsigned digit-serial multiplier modules with partial product or a two's complement digit-serial multiplier modules.

Fast Carry and Carry-Type Select Logic

The DLB provides fast-carry logic that bypasses the ripple-carry interconnect structure for N=4 with a carry look-ahead method. The use of logic modules to make the propagate (P) and generate (G) signal made the DS-FPGA's fast-carry logic very simple. The fast-carry logic greatly increases the efficiency and performance of digit-serial adders/subtractors and multiplier building blocks with N=4.

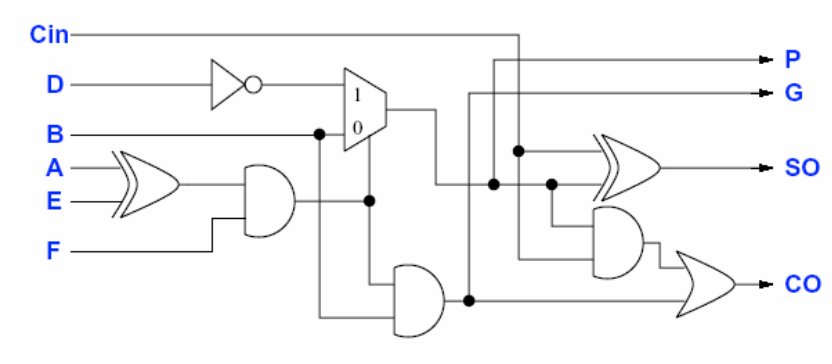

Figure 27: Structure of a Logic Module

As shown in figure 26(b), there is 3:1 MUX gate after four logic module (DLB). This structure comprise of carry-type select logic which is used to select either the ripple-carry chains, the fast-carry logic or the carry-save array. The ripple-carry chains sequentially connect all the LMs in a DLB, supporting N=2, 3 and 4 digit-serial DSP structures. The carry from a lower-order bit moves forward to the higher-order bit via the carry chain.

Table 1: Logic functions which can be implemented by the logic module

Function Name	Function at SO	Function at CO	Input Pattern (A B C D E F)
Full-Adder	$X \oplus Y \oplus Z$	$X \cdot Y + Z \cdot (X+Y)$	X Y Z Y 0 1
Subtractor	$X \oplus Y' \oplus Z$	$X \cdot Y' + Z \cdot (X+Y')$	Y X Z X 1 1
Multiplier Cell	$(X \cdot T) \oplus Y \oplus Z$	$(X \cdot T) \cdot Y + Z \cdot ((X \cdot T) \cdot Y)$	X Y Z Y 0 T
2's comp. Cell	$(X \oplus S) \cdot Y$	0	X 0 0 0 S Y
AND2	$X \cdot Y$	$X \cdot Y'$	X 0 X Y 0 1
NAND2	$(X \cdot Y)'$	Y	X Y X 0 1 1
OR2	$X + Y$	$(X + Y)'$	X 0 1 Y 1 1
NOR2	$(X + Y)'$	0	Y 0 0 X 1 1
XOR2	$X \oplus Y$	$X \cdot Y$	0 X Y 0 0 1
XNOR2	$(X \oplus Y)'$	$X' \cdot Y$	0 0 Y X 1 1
MUX 2:1	$X \cdot S' + Y \cdot S$	$X \cdot S + X'Y'S$	S X S Y 0 1
MUXB 2:1	$X' \cdot S' + Y' \cdot S$	Y	S Y S X 1 1

Register Array

The logic block is a register rich architecture which makes possible a high degree of pipelining, leading to increased performance. The register array contains nine D flip-flops that may be used to register the outputs. The flip-flops share a common clock (CK), clock enable (CE), and set/reset (SR) and they can be set/reset locally or globally. Alternatively, the inputs (REGIN(3:0)) can be used as a direct inputs to the registers. Therefore, the registers can be used independently, if desired.

Overall Evaluation

The architecture proposed above has not been implemented and no elaborate routing structure is defined. The comparisons have been carried out by hand mapping simple datapaths on the proposed architecture and the Xilinx 4000 series FPGA. No actual results about the efficiency of random-logic implementations on the proposed logic block have been given. From the given details of the logic block it can be observed that due to the extremely fine-grained nature of logic modules, the implementation of random logic will need to be distributed among many logic blocks, resulting in large routing, poor logic utilization and maybe inefficient implementation.

6. COMPARISON OF THE APPROACHES FOR DSP SUPPORT

Besides the approaches presented in the previous sections, the idea of reconfigurable computing for DSP has also been implemented by coupling of a programmable digital signal processor (PDSP) along with a reconfigurable fabric. This is idea is proposed in [31] and demonstrated in [2] by a working implementation. The reconfigurable logic block architecture used here resembles that of GARP [Haus97] and is composed of logic blocks each having 2 16x2 LUTs along with an elaborate dedicated carry handling logic structure. The array is organized in a row formation for supporting pipelined applications. The structure called PiCoGA and is shown in figure 28. This structure has been designed to accelerate kernels having large amount of fine grained parallelism while the implementation of wide MACs, multipliers and variable length shifters is relegated to the processor itself.

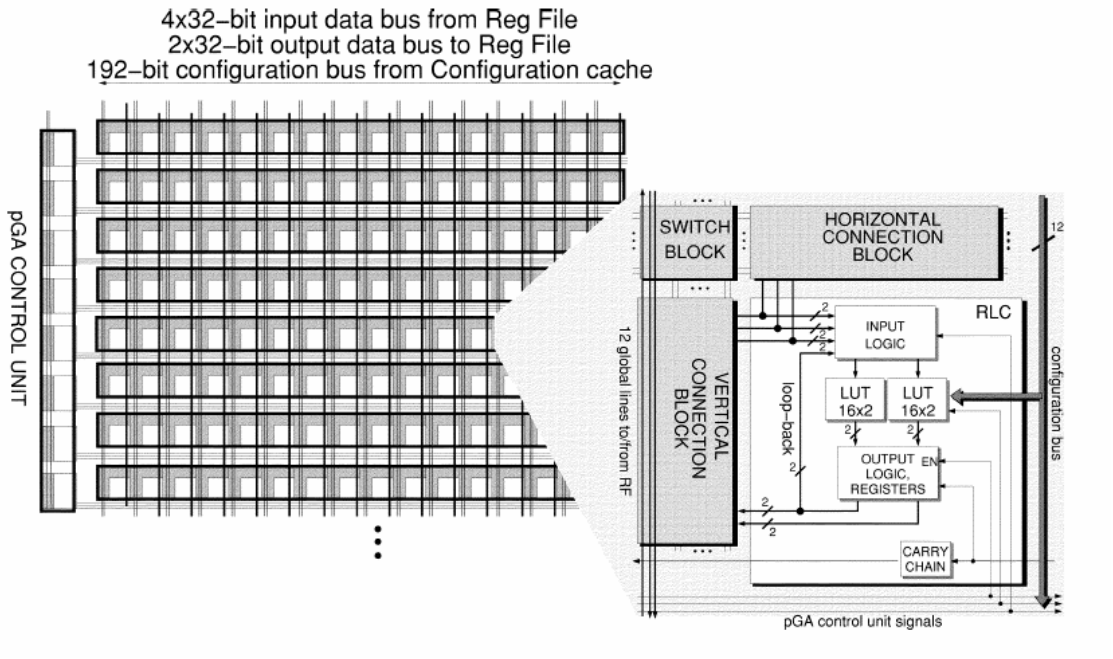

Figure 28: The PiCoGA architecture coupled with a VLIW processor.

The discussion presented in the previous sections highlights the fact that why traditional FPGA architectures are not efficient at supporting DSP oriented applications. On the other hand, the coarse-grained ALU like architectures can achieve better mapping efficiencies for datapath computations but because of their inflexibility, implementing efficient control structures is not directly supported. The use of dedicated coarse-grained blocks (e.g. multipliers) increases

performance but also increases the cost and may constrain application mapping process. Programmable digital signal processors coupled with reconfigurable logic can lead to a balanced solution, and this idea has been only recently demonstrated by a working implementation. Logic blocks with DSP specific tuning, efficiently supporting both the datapath and random logic implementations seem a promising solution to increase the efficiency of FPGAs.

In spite of this some clear advantages of certain approaches, it cannot be said still that a single approach is the best for all application requirements. Each of the approaches to tune the resources for digital signal processing computations has its own suitability with respect to the application demands. Highly computationally intensive applications with large amounts of parallelism can use platform FPGAs, benefiting from the dedicated resources available. Also, here power consumption is usually not an issue and the overall cost/function implemented will be low. Higher flexibility and lower cost can be achieved with logic blocks with DSP specific enhancements and flexibility to implement control logic in an efficient manner. And lastly, field programmable logic based co-processors can benefit from coarse grained blocks where most control functions are implemented by the PDSP itself.

7. CONCLUSION

This report focused on DSP specific application enhancements applied to the design of an efficient logic block of an FPGA. Various approaches were presented, surveying the current state of DSP specific and, in general datapath oriented reconfigurable logic block architectures design. Finally, some observations were presented about the suitability of each approach to the application demands.

BIBLIOGRAPHY

1. Katarzyna Leijten-Nowak, Jef L. van Meerbergen, "***An FPGA Architecture with Enhanced Datapath Functionality***", International Symposium on Field Programmable Gate Arrays, February 2003, pp.195-204

2. Andrea Lodi, Mario Toma, Fabio Campi, Andrea Cappelli, Roberto Canegallo, and Roberto Guerrieri," ***A VLIW Processor With Reconfigurable Instruction Set for Embedded Applications***", IEEE Journal of Solid-State Circuits, vol. 38, no. 11, November 2003, pp.1876-1885

3. A. Ohta, T. Isshiki, T. Watanabe,T. Nakada, K. Akahane, I. Sisla, A. Di, H. Kunieda," ***High Density Bit Serial FPGA with LUT Embedding Shift Register Functionality***", IEEE APCCAS '02, Vol. 1, October 2002, pp. 475-480

4. H. Lee, G. E. Sobelman, "**Digit-Serial Reconfigurable FPGA Logic Block Architecture",** IEEE Workshop on Signal Processing Systems, October 1998, pp. 469-478

5. Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, Brad Hutchings, "***A Reconfigurable Arithmetic Array for Multimedia Applications***", International Symposium on Field Programmable Gate Arrays, February 1999

6. Alireza Kaviani, Daniel Vranesic and Stephen Brown, "***Computational Field Programmable Architecture***", IEEE Custom Integrated Circuits Conference 1998, pp.12.2.1-12.2.4

7. Xilinx. Virtex-II Pro Platform FPGAs. Data sheet. Xilinx, 2004

8. Altera. Stratix-II Handbook. Altera, 2004

9. Andy Gean Ye, "**Field-Programmable Gate Array Architectures and Algorithms Optimized for Implementing Datapath Circuits**", PhD. Dissertation, Electrical and Computer Engineering, University of Toronto, January 2004

10. K. Compton, S. Hauck. Totem, "**Custom reconfigurable array generation**", Proc. of IEEE Symposium on FPGAs for Custom Computing Machines. IEEE, April 2001

11. [Chen92] D. Chen and J. Rabaey, "**A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic-Specific High-Speed DSP Data Paths**", *IEEE Journal of Solid-State Circuits,* December 1992, pp.1895–1904.

12. [Taka98] T. Miyamori and K. Olukotun, "**REMARC: Reconfigurable Multimedia Array Coprocessor**," Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, April 1998.

13. [Wain97] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M.Frank, P.Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "**Baring It All to Software: Raw Machines**," IEEE Computer, September 1997, pp.86–93

14. [Yeun93] A. Yeung and J. Rabaey, "**A Reconfigurable Data Driven Multi-Processor Architecture for Rapid Prototyping of High Throughput DSP Algorithms**," Proceedings of the HICCS Conference, January 1993, pp.169–178.

15. [Also00] A. Alsolaim, J. Starzyk J. Becker, and M. Glesner, "**Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems**," Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, April 2000, pp.205–214.

16. [Bitt96] R. Bittner, P. Athanas, and M. Musgrove, "**Colt: An Experiment in Wormhole Run Time Reconfiguration**," Proceedings of the Conference on High-Speed Computing, Digital Signal Processing, and Filtering Using reconfigurable Logic, 1996.

17. [Gold00] S. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. Taylor, "**PipeRench: a reconfigurable architecture and compiler**," IEEE Computer, April 2000, pp.70–77.

18. [Ebel96] C. Ebeling, D. Cronquist, P. Franklin, and C. Fisher, "**RaPiD A Configurable Computing Architecture for Compute-Intensive Applications**," Proceedings of the International Workshop on Field-Programmable Logic and Applications, 1996.

19. [Mars99] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings, **A reconfigurable arithmetic array for multimedia applications**," Proceedings of the ACM SIGDA International Symposium on Field Programmable Gate Arrays, 1999, pp. 135–143.

20. [Haus97] J. Hauser and J. Wawrzynek, "**Garp: A MIPS Processor with a Reconfigurable Coprocessor**," Proceedings of the IEEE Symposium of Field-Programmable Custom Computing Machines, April 1997, pages 24–33.

21. D. Cherepacha and D. Lewis. **DP-FPGA: An FPGA architecture optimized for datapaths**. VLSI Design, 4(4):329–343, 1996.

22. Alireza Kaviani, "**Novel Architectures and Synthesis Methods for High Capacity Field Programmable Devices**," Ph.D. dissertation in progress, 1998.

23. M. Agarwala and P. Balsara. **An Architecture for a DSP Field-Programmable Gate Array**. IEEE Transactions on VLSI Systems, 3(1):136–141, March 1995

24. N. Miller and S. Quigley. **A novel Field Programmable Gate Array architecture for high speed processing**. In Proc. of Field-Programmable Logic and Applications Conference, pp. 386–390, September 1997.

25. T. Stansfield. **Wordlength as an architectural parameter for reconfigurable computing devices**. In Proc. of Field-Programmable Logic and Applications Conference, pp. 667–676, September 2002.

26. K. Leijten-Nowak and J. L. van Meerbergen. **Embedded reconfigurable logic core for DSP applications**. In Proc. of Field-Programmable Logic and Applications Conference, pp. 89–101, September 2002.

27. K. Leijten-Nowak, A. Katoch. **Architecture and implementation of an embedded reconfigurable logic core in CMOS 0.13µm**. In Proc. of 15th IEEE ASIC/SOC Conference. IEEE, September 2002.

28. J. Rabaey. **Digital Integrated Circuits. A Design Perspective**. Prentice Hall, 1996.

29. G. D. Michelli. **Synthesis and Optimization of Digital Circuits**. McGraw-Hill, Inc., 1994.

30. [Betz99a] V. Betz, J. Rose, and A. Marquardt, **Architecture and CAD for Deep-Submicron FPGAs**, February 1999, Kluwer Academic Publishers.

31. Paul Graham and Brent Nelson, "**Reconfigurable Processors for High-Performance, Embedded Digital Signal Processing**"

32. Altera. FLEX 10KE Programmable Logic Device Family. Data sheet. Altera, 2000.

33. Altera. Stratix Programmable Logic Device Family. Data sheet. Altera, 2002.

34. Atmel. 5K-50K Gate FPGA with DSP Optimized Core Cell and Distributed FreeRAM. Summary. Atmel, 1999.

35. Xilinx. XC4000E and XC4000X Series Field Programmable Gate Arrays. Data sheet. Xilinx, 1999.

36. Xilinx. Virtex 2.5V Field Programmable Gate Arrays. Data sheet. Xilinx, 2000.