Programmable Gate Array Architecture

Bharti Gupta

Research Scholar, CMJ University, Shillong, India

----------♦----------

OVERVIEW

Field-programmable gate arrays are ideal for adaptive systems, since they are reconfigurable and can be programmed to implement any digital logic. Applications of such FPGA-based adaptive systems include face image recognition, on-line failure recovery, and analysis of firefly synchronization.

The main drawback of FPGAs is that they are less efficient than application-specific integrated circuits (ASICs) due to the added circuitry needed to make them reconfigurable. In a recent study , FPGAs are estimated to be 3-4 times slower, 5-35 times larger, and 7-14 times less energy efficient than ASICs depending on the application and the flexibility of the FPGA.

Traditionally, FPGA research focused on reducing the speed and area overhead [9]. In recent years, however, much of the focus has shifted to improving the energy efficiency. This shift is due to process scaling and increased demand for low-power applications. Although process scaling reduces the energy needed to perform a given computation (since wires and transistors are smaller), it increases power dissipation per unit area and therefore the overall power for a given die size . At the same time, demand for low-power applications is increasing due to the proliferation of hand-held devices and increasing energy costs. For hand-held and other battery operated devices, reducing power increases battery life. For non-mobile devices, reducing power consumption lowers operating, packaging, and cooling system costs.

There are many ways to make FPGAs more energy efficient. The various techniques can be divided into five categories: process, circuit, architecture, system, and computer-aided design (CAD). Process techniques refer to the use of new low-power process technologies offered by the semiconductor manufacturers. Circuit techniques refer

to the transistor-level implementation of the logic and routing resources. Architecture techniques refer to functionality of the logic, I/O, and memory resources and the connectivity between these resources. System techniques refer to high-level low-power techniques such as dynamic voltage control, turning resources off when they are not being used, and run-time reconfiguration. Finally, CAD refers to enhancements made to the mapping tools which are used to configure the FPGA.

1.1 Basic Components and Blocks

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to

set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow.[3][4] Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip.[5] Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

FPGAs are versatile configurable electronic components that are utilized in accelerators to implement tailored computational logic specific to the application being executed. In the hybrid computer system, the FPGA acts as a configurable co-processor to a CPU, allowing applications to use it as an application-specific hardware accelerator for computationally intensive tasks. Depending on the application, this can provide orders of magnitude faster execution as well as radically reduced power consumption. The FPGA component can be reconfigured for new applications, making it possible to utilize the hybrid computer system for a wide range of tasks.

FPGAs can be used to build digital logic circuitry – including tailor made co-processors for a particular algorithm to be used as part of a hybrid computing system. Even though the P in FPGA stands for Programmable, it is important to realize that this is not in the software sense of programmable. For an FPGA it simply means that a circuit design can be loaded.

The bulk of an FPGA is made up from a large number of identical configurable logic blocks, CLBs. Each CLB consists of a number of slices which in turn consist of a number of (typically two or four) logic cells that can be configured to perform basic logic functions (such as and, or, not) on digital signals using the lookup table, LUT. The CLBs are interconnected through programmable switch matrixes, PSM, to form units that perform more complex functionality.

FPGAs also provide internal RAM memory banks and specialized multiplier logic blocks, Multiply-accumulate circuits, MACs, for efficient multiplication and addition. FPGAs may have other block with specialized functions to serve purposes such as digital signal processing. Finally, each I/O pin of the FPGA device can be programmed to provide the electrical interfaces needed to connect the FPGA to the system it is part of.

The number of logic cells, sizes and number of internal RAM memory banks, and MACs all vary between different FPGAs. FPGAs that are used in hybrid computer systems typically have about 100k – 300k logic cells, 500kB of internal RAM and 100 MACs. The FPGAs can have almost 1000 configurable I/O pins, that are used in hybrid computers to provide interfaces to the host system, as well as to locally attached memory directly accessible to the FPGA.

Altium Designer provides a generic set of FPGA macro components – symbolic representations of blocks of functionality that a user desires to add to an FPGA design. These components are presented to the user as FPGA-ready schematic symbols (or graphical representations in an OpenBus System) that can be instantiated into a design. FPGA-ready schematic components are like traditional PCB-ready components, except instead of the symbol being linked to a PCB footprint, each is linked to a pre-synthesized EDIF model.

The pre-synthesized components are supplied as object code entities without having to expose underlying RTL- or netlist-level source code. The system includes multiple libraries providing a comprehensive set of pre-synthesized components, ranging from simple gate-level functional blocks, up through high-level hardware functions, such as multipliers and pulse-width modulators, to high-level functions, such as processors and communications peripherals.

These components can be instantiated into designs by the system user and then the whole design can be targeted to a suitable physical device. Altium Designer automatically manages the resources required to instantiate the design in the chosen FPGA device, by ensuring that the EDIF models specific to that device are correctly chosen and linked to the generic symbols placed when capturing the design.

A Swappable Logic Unit (SLU) is an FPGA-based logic circuit that can be managed by a virtual hardware operating system. Since the SLU concept is still in its infancy, it is unreasonable to expect circuit design systems to be enhanced to include SLU awareness at this stage. Therefore, this section describes a tool that can be used as a back end to any XC6200 circuit design system. It analyses XC6200 circuitry, to detect SLUs and then collect information about their external interfaces. This information is needed by an operating system so that it can manage the SLUs when they are in use.

1.2 Architectural Styles

The field-programmable gate array (FPGA) is a revo¬lutionary idea in semicustom integrated circuits that reduces the IC manufacturing time from months to min¬utes and prototype cost more than three decades. The FPGA was introduced in and newer versions have been presented in [2]-[9]. It is similar to a gate array in structure, but can be field-programmed to specify the function of the logic blocks and their interconnection. As a result of the programmability, the architecture of an FPGA is more complex than that of a conventional gate array.

In this section wc focus on the logic block architecture, and study the effect of logic block functionality on FPGA area. We ignore speed considerations, even though they are very important, because wc need first to determine the plausible architectures from an area perspective. In later studies we can use this information to know the cost, in area, of obtaining an FPGA with a particular perfor¬mance. An associated study investigates the effect of routing structure flexibility on routability of FPGA's .

The logic block functionality is an important factor in the FPGA architecture. It can be loosely defined as the number of logic blocks required to implement an (un¬specified) set of circuits. A precise and usable definition of functionality remains an open question. If the block has insufficient functionality then too much area must be devoted to the interconnection. If the block has excess functionality then it may suffer from underutilization and wasted active area.

In this section wc focus on a simple logic block architec¬ture that contains a /C-input lookup table to implement combinational logic and a D flip-flop. Wc address two questions concerning this block with the following results:

- What is the best number of inputs, K, to use for the combinational function? Note that K is direct mea¬sure of functionality. Our results show that the best number of inputs is consistently between three and four, and is almost the same whether or not the block contains a D flip-flop.

- Should the logic block contain a D flip-flop? Experi¬ments indicate that the presence of a D flip-flop in the logic block always reduces chip area.

The explanation of these results lead to an important insight into the functionality-area trade-off of FPGA's. First, we find that interconnection area completely domi¬nates active area so that the trade-off between routing area and logic block functionality is the key one. Second, as intuition suggests, when the functionality of the logic block increases, the total number of logic blocks required to implement a circuit decreases. However, when func¬tionality increases, so does the number of pins on the logic block. Wc have found that the total routing area is a direct function of the number of connected pins on the logic block: as the number of pins increases, the routing area increases significantly. Therefore, a beneficial in¬crease in functionality of the logic must reduce the total number of blocks to more than compensate for the in¬creased routing area. This point further implies that a good choice for an arca-efficient block is one that has high functionality per connected pin.

The architectural choices that affect the area of an FPGA depend on the programming technology, which is the underlying method by which logic functions are con¬figured and routing connections arc made. For example, the programming technology used in creates logic functions using static RAM lookup tables, and performs routing using pass transistors and multiplexors. The FPGA described in uses an antifusc for both logic configura¬tion and interconnection that, when blown, causes two metal tracks to be electrically joined. The FPGA de¬scribed in uses an FPROM programming technology. The experiments described in this section account for the area requirements of differing programming technologies.

Previous work on FPGA's has taken the form of de¬scriptions of a specific architecture and implementation. The first dcvicc that can be considered a FPGA was introduced in 19X6 , and subsequent devices are de¬scribed in AND. The programming technology is based on static RAM bits that are loaded into the chip when the system is turned on. The most recent logic block architecture consists of a combinational block followed by two rescttablc D flip-flops. The combinational block is a static RAM lookup table that can be configured to realize any five-input one-output logic function, or any two four-input one-output logic functions in which the inputs must be selected from the same set of signals.

The global routing architecture is similar to a chan¬neled gate array. Connections are made from the logic block to the channel via multiplexors controlled by static RAM. At the intersection of horizontal and vertical chan¬nels (the switchbox). connections arc made by transistors that are turned on or off using static RAM bits. There are dedicated local interconnects between neighboring blocks that are not switched, as well as "long" lines that traverse entire channels. Dedicated lines for global clock distribu-tion are also present.

An FPGA based on antifuse programming technology was introduced in [4]-[6]. The logic block consists of three multiplexors and a logic OR gate, which can be connected in various ways using the antifuse to perform a wide range of combinational and sequential logic func¬tions. The

interconnection architecture consists of horizontal channels with staggered segments of wires that can be joined by antiques. and vertical connections across the logic blocks. A global clock distribution scheme is also included.

A third FPGA architecture, based on an EPROM programming technology, was introduced in [7], The logic block architecture is very similar to that in a single PLD:

Two-level AND-OR logic followed by a single D flip-flop. An additional level of logic is provided by an expanded product-term array associated with a group of logic blocks. Inputs are wired-OR selected from a bus using programmed transistors. The interconnection scheme is a two-level hierarchy. A bus structure connects logic blocks within one level of the hierarchy and between groups of logic blocks at the second level of the hierarchy.

Two more FPGA architectures were recently introduced. The first, based on a static RAM programming technology 18], has a logic block consisting of a two-input NAND gate, a multiplexor, and a latch. Routing is performed using the same logic element so that routing and logic arc directly traded off. The second recent FPGA uses an EPROM programming technology and a logic block based on AND-OR gates similar to a PLD. Interconnection is done with a two-level hierarchy of buses.

Gray and Kean have also proposed an FPGA-like Structure. It uses a static RAM programming technology and a logic block based on interconnections of five multiplexors. The interconnection scheme is two connections to each nearest neighbor, one in each of the four orthogonal directions. This chip has been used to design an encryption algorithm and a fluid flow simulator.

In this section we explore a range of FPGA logic block architectures rather than defining a single complete architecture as above. This provides insight into the trade-offs involved in choosing a logic block, rather than a "point" experience. To make this first exploration plausible, we investigate only the effect of architectural decisions on the area efficiency. To our knowledge, this is the first such study. An early version of this work appeared in [13].

This section is organized as follows. Section details our experimental approach to answering the questions raised above, and gives the architectural model used in the experiments. Section presents the results of the experiments and the detailed reasoning, both theoretical and experimental, for these results. Section draws more general conclusions from the specific results of the experiments.

This section reviews the basic structure of an FPGA, focusing on what makes FPGAs power-hungry. FPGAs are made up of a large number of configurable logic blocks, which implement the logic part of digital circuits, and a configurable routing fabric, which implements the connections between the logic blocks. Modem FPGAs also have embedded fixed logic components, such as memories and arithmetic logic units. These embedded components are typically aligned with the logic tiles, and are often arranged in rows or columns. Figure provides an abstract view of an FPGA with programmable logic and embedded fixed-funet ion components.

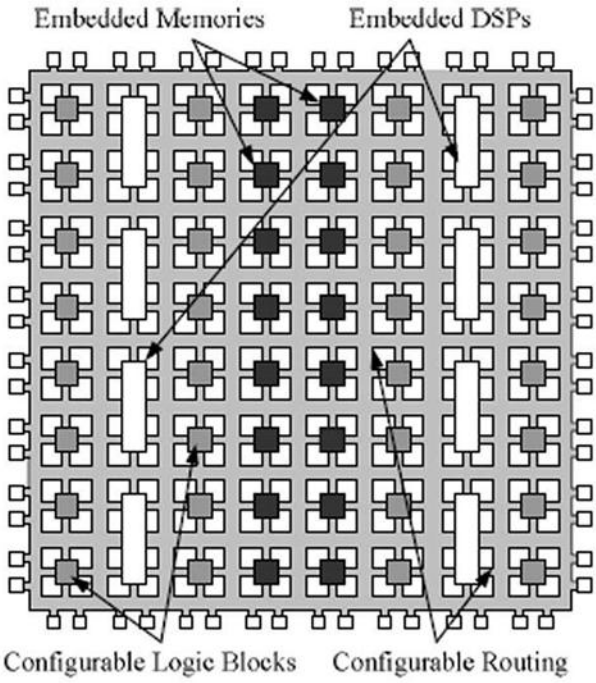


Figure: A generic FPGA with embedded components.

FPGAs dissipate more static power than ASICs for a number of reasons. FPGAs use a large amount configuration memory to control every programmable routing switch and logic function in the FPGA. Each configuration bit dissipates static power. Another reason is that the programmable logic blocks are implemented using lookup-tables, which have significantly more transistors than the corresponding logic gates in an ASIC. Similarly, FPGA routing resources use significantly more transistors than in ASICs because of the large number of multiplexers needed to make the routing flexible.

FPGAs also dissipate more dynamic power than ASICs. In both an ASIC and FPGA, connections between gates are associated with some amount of parasitic capacitance due to the metal wire used to implement the connection as well

as the driver and driven transistors. However, as described above, a connection in an FPGA also contains a large number of programmable switches. These switches significantly increase the parasitic capacitance on the wire segments and charging and discharging this parasitic capacitance consumes dynamic power.

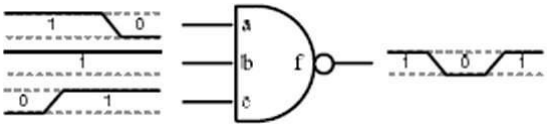
Figure shows a breakdown of core power consumption in a commercial 90-nm FPGA [61]. The figure shows that the routing resources dissipate the greatest amount of power, followed by logic and clock network resources. This study also reports that dynamic power accounts for 62% of the total power, while static power accounts for 38%. There is also recent work which considers FPGAs with embedded memories; such memories are found to account for 14% of core dynamic power [56].

The architecture and the circuit-level implementation of the FPGA is key in reducing power, since it directly affects the efficiency of mapping applications to FPGA resources, and the amount of circuitry to implement these resources.

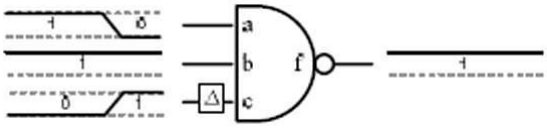
A number of studies have investigated low-power FPGA architecture design. Energy-efficient FPGA routing architectures and low-swing signalling techniques to reduce power are described . In [55], a new FPGA routing architecture that utilizes a mixture of hardwired and traditional programmable switches is proposed, which reduces static and dynamic power by reducing the number of configurable routing elements. In [7], a novel FPGA routing switch with high-speed, low-power, or sleep modes is presented The switch reduces dynamic power for non timing critical logic and standby power for logic when it is not being used. In , power-gating is applied to the switches in the routing resources to reduce static power; duplicate routing resources, that use either high or low Vdd, are used to reduce dynamic power. In [30], energy-efficient modules for embedded components in FPGAs are introduced to reduce power by optimizing the number of connections between the module and the routing resources, and by using reduced supply voltage circuit techniques. In [27], several power reduction techniques, such as register file elimination and efficient instruction fetch, are proposed for a coarse-grain reconfigurable cell-based architecture; up to 3.6 times lower energy than an ARM7 device, and up to 6 times lower energy than a C55X DSP, is reported.

Although significant improvements have already been made, many opportunities to further reduce power in FPGAs remain. The rest of this section describes two recent improvements: minimization of FPGA glitch power, and efficient FPGA clock network design.

The first improvement concerns FPGA glitch reduction. Glitching occurs when values at the inputs of a LUT toggle at different times due to uneven propagation delays of those signals. If the arrival times are far enough apart, spurious transitions can be produced at the LUT output, as shown in Figure . A recent study suggests that glitching accounts for 31% of dynamic power dissipation in FPGAs [33].



(a) Original circuit with glitch



(b) Glitch removed by delaying input c

Figure: Example of delay insertion to eliminate glitching.

The study proposes a method for minimizing glitching which involves adding configurable delay elements to the inputs to each logic element in the FPGA. After place and route, detailed tuning information is used to configure these delay elements so as to align the arrival times at the inputs of each logic element. This eliminates glitches as long as the arrival times can be aligned closely enough, as shown in Figure .

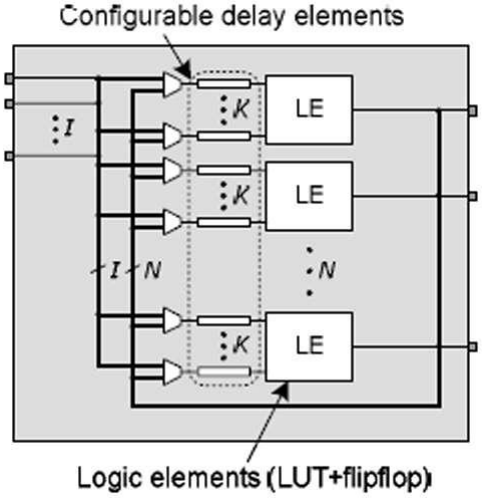


Figure: FPGA logic block with configurable delay elements.

The amount of glitching that can be eliminated depends on several factors. Specifically, the resolution, maximum delay, location, and amount of the programmable delay

elements all have an affect on glitch elimination and overhead. It was found that, on average, the proposed technique eliminates 87% of the glitching, which reduces overall FPGA power by 17%, while the added circuitry increases the overall FPGA area by 6% and critical-path delay by less than 1%.

A 17% reduction in power is significant. Moreover, the method can be applied to all commercial FPGAs, and requires only minor changes to the CAD flow or the rest of the architecture. The gains are roughly independent of those that can be obtained using process enhancement techniques. However, there may be some overlap in these gains with those that can be obtained using a power-aware CAD flow, since by reducing the activity of high-activity signals, there may be less "low-hanging fruit" available for the power-aware CAD flow.

The second recent improvement concerns low-power clock network design. New FPGAs are sophisticated enough to implement large system-level applications. These applications often have many clock domains. As an example, consider a communications application connected to several I/O ports. Each port might have its own clock, meaning the circuitry connected to each port must be controlled by a separate clock. FPGA vendors support such applications through the use of programmable clock networks that are flexible enough to support a wide range of applications, yet have low skew.

1.3 Triptych

The FPGA architecture we present in this section differs from other FPGAs by matching the structure of the logic array to that of the target circuits, rather than providing an array of logic cells embedded in a general routing structure. By matching the physical structure to the logical structure, we reduce the amount of “random” routing that is otherwise required.

Figure shows a high-level view of a typical multi-level combinational logic circuit. The flow is shown as unidirectional, from inputs to outputs. From the point of view of each input, the data flow forms a fanout tree (shown with solid arrows) to those outputs that the input affects. From the point of view of each output, the data flow forms a fanin tree (shown with dashed arrows) from those inputs it depends upon. It is this fanin/fanout tree form that Triptych emulates architecturally by arranging RLBs into columns, with each RLB having a short, hard-wired connection to its nearest neighbors in adjacent columns.

The basic structure is augmented with segmented routing channels between the columns that facilitate larger fanout structures than is possible in the basic array. Finally, two copies of the array, flowing in opposite directions, are overlaid. Connections between the planes exist at the crossover points of the short diagonal wires. It is clear that this array does not allow arbitrary point-to-point routing like that associated with Xilinx and Actel FPGAs. However, we claim that this array matches the form of a large class of circuits and that mapping will produce routable implementations.

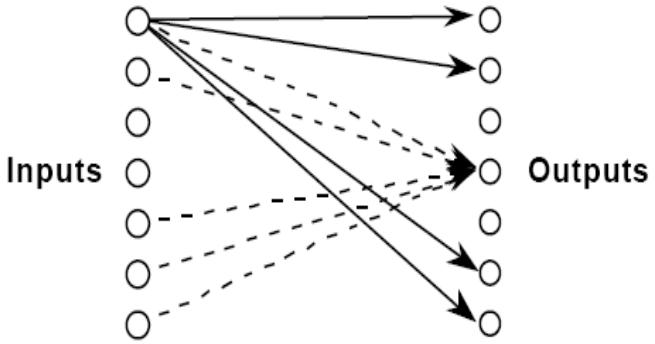


Figure View of a multi-level combinational logic circuit as nterleaved fanin/fanout trees.

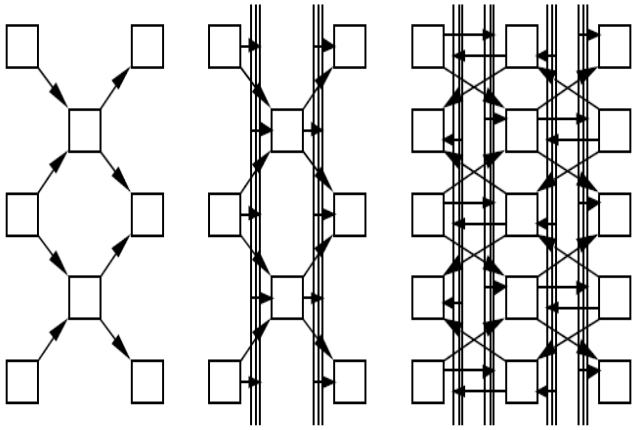


Figure The overall structure of the Triptych FPGA shown in a progression of steps highlighting more and more features.

The basic fanin/fanout structure on the left isaugmented with segmented routing channels that make a third input and a third output available to the RLBs. The structure on the right is obtained by merging two copies of the middle structure, with data flowing in opposite directions in the two copies. Not shown are the connections between the two copies, which permit internal feedback.

Each RLB in the array has three inputs and three outputs and may perform an arbitrary logic function of the three inputs, with the result optionally held by a master/slave D-latch (Rose 1990). Routing in the Triptych array is in three forms: horizontally through the RLBs (by selecting an input to be routed to an output), diagonally through short wires to neighbors, and vertically through the segmented channels between columns of RLBs. Only one input and one output can be connected to the vertical wires; the other two must be on the local diagonal interconnect.

Circuits can be mapped onto this array by partitioning the logic into circuit DAGs containing nodes with at most three inputs. These DAGs are then mapped to the physical structure, with the inputs at one side of this structure and the outputs generated at the other.

The nodes of the DAGs are placed such that input signals are available from the neighbor nodes or along a vertical connection. As Rose suggests in (Singh et al 1990), delay can be minimized by using mostly direct, hard-wired connections for the critical path. Triptych implementations do not strive for 100% logic utilization. Many RLBs will be used to provide routing, either to fanout a signal or to pass it forward to the next level. Sometimes a mapping will leave some cells unused to achieve a routable placement of nodes. Examples are provided below.

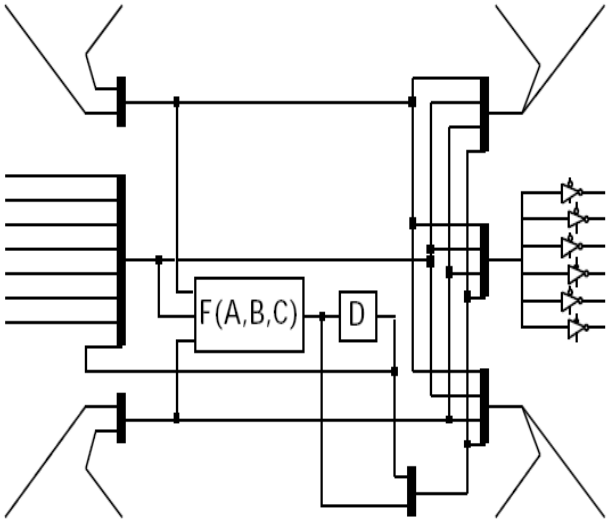


Figure. Triptych RLB design. The RLB consists of: 3 multiplexers for the inputs, a 3-input function block, a master/slave D-latch, a selector for the latched or unlatched result of the function, and 3 multiplexers for the outputs.

REFERENCES

[1] Ahrens, M., Gamal, A., "An FPGA Family Optimized for High Densities and Reduced Routing Delay,"Actel Corporation, IEEE Custom Integrated Circuits Conference, 1990.

[2] Atmel "Configurable Logic Design and Application Handbook" , 1995.

[3] Black, P, Meng. T., "A 140 mb/s 32 State, Radix 4 Viterbi Decoder," IEEE Journal of Solid State Circuits Vol. 27, No. 12, December 1992, pp. 1877-1885.

[4] Bowhill, W., et al, "A 433 MHz 64b Quad Issue RISC Microprocessor," IEEE Journal of Solid State Circuits, Vol 31, No. 11, November 1996, pp ##.

[5] Brebner, G. , "Configurable Array Logic Circuits for Computing Network Error Detection Codes," Journal of VLSI Signal Processing, 6, 1993, pp. 101-117.

[6] Chandrakasan, A. , "Low Power Digital CMOS Design," PhD. Thesis, U.C. Berkeley, August 1994.

[7] CLAy Family Introduction Datasheet, National Semiconductor, June 1994.

[8] DeHon, A. , "Reconfigurable Architectures for General Purpose Computing," M.I.T. PhD Thesis, A.I. Technical Report 1586, October 1996.

[9] Dobbelaere, I. , Horowitz, M. , Gamal, A. , "Regenerative Feedback Repeaters for Programmable Interconnections", ISSSC Digest of Technical Sections 1995, p.116- 117.

[10] Farrhi, A. , Sarrafzadeh, M. , "FPGA Technology Mapping for Power Minimization," International Workshop on Field-Programmable Logic and Applications, FPL '94. Proceedings, Springer-Verlag, 1994. p. 66-77.

[11] Gamal, A. , et al., "An Architecture for Electrically Configurable Gate Arrays," IEEE Journal of Solid-State Circuits, Vol. 24, No. 2, April 1989, pp 394-398.

[12] George, V. , The Effect of Logic Block Granularity on Interconnect power in a Reconfigurable Logic Array", CS 294 report, May 1997.

[13] Goto, G., et al "A 4.1ns Compact 54x54 Multiplier Utilizing Sign-Select Booth Encoders," IEEE Journal of Solid State Circuits, Vol 32, No. 11, November 1997, pp 1676-1683.

[14] Hauck, S., Borriello, G., Ebeling, C. , "Triptych: An FPGA Architecture with Integrated Logic and Routing", in Advanced Research in VLSI and Parallel Systems:

Proceedings of the 1992 Brown/MIT Conference, pp. 26-43, March 1992.

[15] Infopad Project, U.C. Berkeley, http://infopad.EECS.Berkeley.EDU/infopad

[16] Izumikawa, M. , et al., "A 0.25um CMOS 0.9v 100-MHz DSP Core," IEEE Journal of Solid-State Circuits, Vol. 32, No. 1, January 1997, p. 52-60.

[17] Jou, S., et al, "A Pipelined Multiply-Accumulator using a High-Speed Low-Power Static and Dynamic Full Adder Design," IEEE Journal of Solid State Circuits, Vol 32, No. 11, November 1997, pp ##.

[18] Kaushik, R. Prasad, S. , "FPGA Technology Mapping for Power Minimization," International Workshop on Field-Programmable Logic and Applications, FPL '94. Proceedings, Springer-Verlag, 1994. p. 57-65.