FPGA Architecture*

1st t Evrard Leuteu Feukeu
Hochschule Hamm- Lippstadt
Electrical Engineering
Hamm, Germany
evrard.leuteu-feukeu@stud.hshl.de

CONTENTS

***Abstract*—Nearly 25 years ago, field programmable gate arrays (FPGAs) were initially made available. They have since grown quickly and are now a common implementation medium for digital circuits. The improvement in process technology has significantly increased the logic capacity of FPGAs, making them an attractive alternative for the execution of more elaborate and substantial designs. Additionally, the fact that their logic and routing resources are programmable has a significant impact on the space, speed, and power consumption of the finished product.**

***Index Terms*—component, formatting, style, styling, insert**

I. INTRODUCTION

Almost any type of digital circuit or system can be created using field programmable gate arrays (FPGAs), which are prefabricated silicon devices. FPGAs offer a more affordable solution for low- to medium-volume productions. compared to application-specific integrated circuits, and a quicker time to market (ASIC), which often need a lot of time and money to develop get the first gadget. FPGAs, on the other hand, need little configuration time and They are priced between a few hundred and a few thousand dollars. Also A component of the FPGA can be partially modified to meet different requirements. An FPGA is still operating in its entirety. Upgrading the final product with any upcoming changes is simple and only requires downloading a new application bitstream. However, the versatility of FPGAs, which is their fundamental benefit, is also one of their biggest disadvantages. FPGAs are substantially bigger, slower, and more powerful due to their flexibility. as compared to their ASIC counterparts.

II. PROGRAMMING TECHNOLOGIES

There are a number of programming technologies that have been used for reconfigurable architectures. Each of these technologies have different characteristics which in turn have significant effect on the programmable architecture. Some of the well known technologies include static memory [51], flash [22], and anti-fuse [25].

A. SRAM-Based Programming Technology

For SRAM-based FPGAs, static memory cells are the fundamental cells used. Static memory (SRAM)-based

programming technology is used in most commercial vendors' products [26, 33]. For flexibility, these devices employ static memory cells that are partitioned over the FPGA. In Fig. 2.1, a sample of one of these memory cells is displayed. The following purposes are the primary uses of SRAM cells in an SRAM-based FPGA:

1. To program the FPGAs' routing interface, which is typically controlled by tiny multiplexors.
2. To program Configurable Logic Blocks (CLBs), which carry out logic operations.

Due to its reprogrammability and use of industry-standard CMOS process technology, SRAM-based programming technology has supplanted other approaches as the preferred method for programming FPGAs. This has resulted in greater integration, faster processing, and less dynamic power consumption of new processes with smaller geometries. SRAM-based programming technology does have some disadvantages, though. For instance, an SRAM cell needs 6 transistors, which makes this technology more expensive to utilize than other programming technologies in terms of area. Additionally, because SRAM cells are volatile, additional devices are needed to permanently store the configuration information. The cost and area overhead of SRAM-based FPGAs are increased by these extraneous devices.

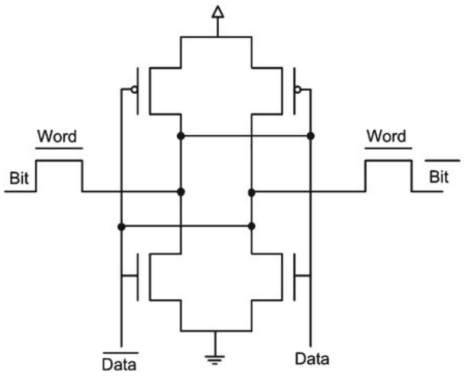

Fig. 1. Static memory cell

B. Flash Programming Technology

Use of flash or EEPROM-based programming technology is an alternative to SRAM-based programming technology. The use of Flash-based programming technology has many benefits. For instance, the nature of this programming technology is nonvolatile. Additionally, the area efficiency of flash-based programming is higher than that of SRAM-based programming. The Flash-based programming technology is not without its drawbacks. Flash-based devices cannot be configured/reprogrammed indefinitely, in contrast to SRAM-based programming technology. Flash-based technology also makes use of an unconventional CMOS technique.

C. Anti-fuse Programming Technology

Anti-fuse programming technology is a substitute for SRAM and flash-based technologies. Anti-fuse programming technology's low area is its main benefit. In comparison to the other two programming systems, this technology also has lower parasitic capacitance and resistance. Furthermore, this technology is inherently non-volatile. However, there are some serious drawbacks to this programming technology. For instance, this technique doesn't employ the common CMOS process. Additionally, devices using anti-fuse programming technology cannot be reprogrammed. This section provides an overview of three programming languages that are often used, outlining their benefits and drawbacks for each. A reprogrammable, non-volatile programming technology that makes use of a typical CMOS process would be ideal. Evidently, none of the technologies discussed above meet these requirements. Although SRAM-based programming technology is the most popular, it is not the only one. The fundamental justification for this is that it makes use of the common CMOS process, and it is anticipated that this technology will continue to outpace the other two programming approaches for this very reason.

III. Configurable Logic Block

A configurable logic block (CLB) is a fundamental part of an FPGA that gives a target application design the essential logic and storage functions. The fundamental component can be either of the following to offer the fundamental logic and storage capability: a processor's core or a transistor. These are the two extremes, though, where at one side In the case of transistors, the fundamental component is extremely fine-grained and requires a significant quantity of programmable connectivity that eventually yields an FPGA has inefficient use of space, poor performance, and high power usage. On the other hand, because the fundamental logic block in a processor is so coarse-grained, it cannot be used to create little functions without wasting resources. There is a range of fundamental logic building pieces that lies between these two extremes, and there is a range of fundamental logic building elements. NAND gate-based logic blocks [34], multiplexor connections [35], lookup tables (LUTs) [36], and wide input gates [34] are a few of them. as seen in fig: 3

IV. FPGA Routing Architechture

As was previously said, an FPGA's programmable logic blocks, which are connected to one another over a programmable routing network, provide the computing functionality. To create any user-defined circuit, this programmable routing network offers routing links between

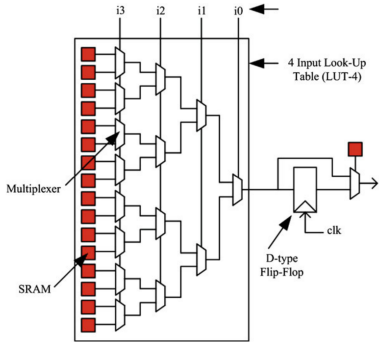

Fig. 2. Logic block element

logic blocks and I/O blocks as seen in fig: 4. An FPGA's routing interface is made up of programmable switches and wires that make the necessary connections. The programmable technology is used to configure these programmable switches. Since FPGA architectures assert that they are candidates for the construction of any digital circuit, they must have very flexible routing interconnects that can support a large range of circuits with a wide range of routing requirements. Despite the fact that each circuit's routing requirements are different, it is possible to build the routing interconnect of a circuit by taking use of certain of these circuits' common qualities. Architecture for FPGA. For instance, the majority of the designs show locality, necessitating several short wires. However, there are also some distant relationships, including results in the requirement for few, long wires. FPGA architectures can be characterized as either hierarchical [38] or island-style [39] based on the overall organization of the architecture's routing resources. We give a thorough description of both routing architectures in this section.

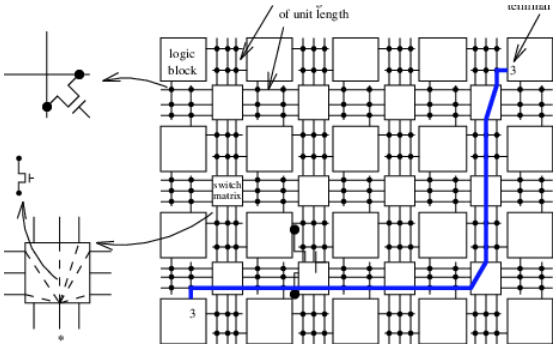

Fig. 3. Routing example

1) Island-Style Routing Architecture:

An island style FPGA architecture is also called as old-fashioned architecture or mesh based FPGA architecture which shown in the Fig.5. The reason it is termed island-style architecture is because customizable logic blocks in this design resemble islands in a sea of routing links. Configurable logic blocks (CLBs) are organized in a 2D grid and connected by a programmable routing network in this architecture. The FPGA chip's Input/Output (I/O) blocks are connected to the programmable routing network as well. The pre-fabricated cable segments and programmable switches that make up the routing network are arranged in horizontal and vertical routing channels. FPGA filled up 80-90logic block is interconnected with the connection boxes (CB).The connectivity of input and output pins with the adjacent routing channel called as FC (in) and FC (out). If FC (in) =1.0 which means all the channel are linked to the input pins. If FC (out) =0.5 which means 50the input pins and it is in track of adjacent routing channel [3,4]. If single-driver directional wiring is used instead of bidirectional wiring, 32delay and 25are logic block arranged in the 2-D mesh style with routing resource which is evenly distributed. of wire contained in a channel which is pre-set during the fabrication and one of the key choices made by the architect. The main pros for the island style architecture are efficient connection for a variety of design net length.

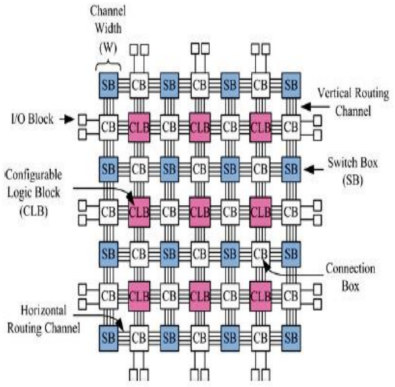

Fig. 4. Island style routing example

2) Hierarchical Routing Architecture:

In hierarchical routing architecture is otherwise called as tree-based architecture. The main goal in hierarchical routing is to increasing the circuit speed and decreasing circuit area. The FPGA logic block divides into groups and clusters. In hierarchical architecture, the signal bandwidth move away from the bottom level and the top level has the widest path of hierarchy. In the lowest levels of routing hierarchy connection between logic block and cluster through wire. The grouping of LBs forms to clusters. The wires are directly linked to the logic block which placed in logic leave of the interconnected tree and all further wire in sections are separated from the logic part. A logic block has 2 pair which is 2-input LUT and D-type flip flop. The output pins are connected fully and the input pins are based on the k-strategy. It has two levels such that compressing and non-compressing [4]. The advantage of hierarchical routing architecture, if interconnection delay is not significant, the delay is almost equal for all connection. It has superior performance for some logic design [1].Due to the fully popular switch pattern, hierarchical offer lower density than other FPGA. It has more secure and too lower delay.

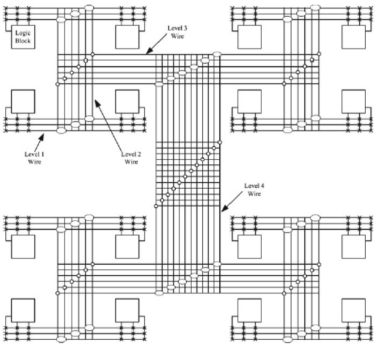

Fig. 5. Hierachical routing example

V. Software Flow

Over the past 20 years, FPGA architectures have been the subject of extensive research. The creation of Computer Aided Design (CAD) tools for mapping applications to FPGAs is a key component of FPGA architectural research. It is commonly known that the The efficiency of an FPGA-based implementation heavily influences its quality. of the related set of CAD tools. advantages of a well-designed feature in general If the CAD tools are unable to utilize rich FPGA architecture, one of the functions that the FPGA offers. Consequently, CAD algorithm research is crucial. in order to make the required architectural advancements, in order to reduce performance gaps between FPGAs and other computing hardware, such as ASICs.

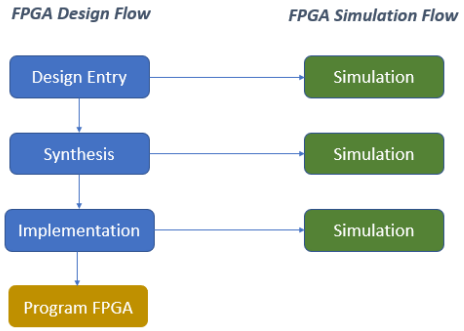

Fig. 6. Software flow

1) Logic Synthesis:

The flow of FPGA starts with the logic synthesis of the netlist being mapped on it. Logic synthesis [7, 8] transforms an HDL description (VHDL or Verilog) into a set of boolean gates and Flip-Flops. The synthesis tools transform the register-transfer-level (RTL) description of a design into a hierarchical boolean network. Various technology-independent techniques are applied to optimize the boolean network. The typical cost function of technology-independent optimizations is the total literal count of the factored representation of the logic function. The literal count correlates very well with the circuit area. Further details of logic synthesis are beyond the scope of this documentation.

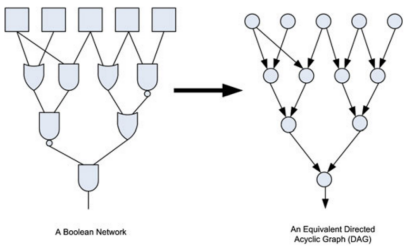

Fig. 7. Acyclic graph representation

2) Technology Mapping:

The output from synthesis tools is a circuit description of Boolean logic gates, flipflops and wiring connections between these elements. The circuit can also be represented by a Directed Acyclic Graph (DAG). Each node in the graph represents a gate, flip-flop, primary input or primary output. Each edge in the graph represents a connection between two circuit elements. Figure 9 shows an example of a DAG representation of a circuit. Given a library of cells, the technology mapping problem can be expressed as finding a network of cells that implements the Boolean network. In the FPGA technology mapping problem, the library of cells is

composed of k-input LUTs and flip-flops. Therefore, FPGA technology mapping involves transforming the Boolean network into k-bounded cells. Each cell can then be implemented as an independent k-LUT. Figure 9 shows an example of transforming a Boolean network into k-bounded cells. Technology mapping algorithms can optimize a design for a set of objectives including depth, area or power. The FlowMap algorithm [17] is the most widely used academic tool for FPGA technology mapping. FlowMap is a breakthrough in FPGA technology mapping because it is able to find a depth-optimal solution in polynomial time. FlowMap guarantees depth optimality at the expense of logic duplication. Since the introduction of FlowMap, numerous technology mappers have been designed that optimize for area and run-time while still maintaining the depth-optimality of the circuit [18-19]. The result of the technology mapping step generates a network of k-bounded LUTs and flip-flops.

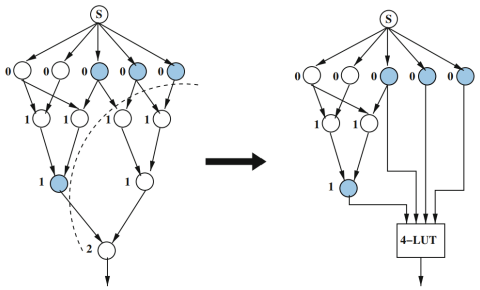

Fig. 8. Technology mapping example

3) Clustering/Packing:

The logic elements in a Mesh-based FPGA are typically arranged in two levels of hierarchy. The first level consists of logic blocks (LBs) which are k-input LUT and flip-flop pairs. The second level hierarchy groups k LBs together to form logic blocks clusters. The clustering phase of the FPGA CAD flow is the process of forming groups of k LBs. These clusters can then be mapped directly to a logic element on an FPGA. Figure 2.20 shows an example of the clustering process. Clustering algorithms can be broadly categorized into three general approaches, namely top-down [12, 21], depth-optimal [22, 28] and bottom-up [5, 13]. Top-down approaches partition the LBs into clusters by successively subdividing the network or by iteratively moving LBs between parts. Depth-optimal solutions attempt to minimize delay at the expense of logic duplication. Bottom-up approaches are generally preferred for FPGA CAD tools due to their fast run times and reasonable timing delays. They only consider local connectivity information and can easily satisfy clusters pin constraints. Top-down approaches offer the best solutions; however, their computational complexity can be prohibitive.

4) Placement:

Placement algorithms determine which logic block within an FPGA should implement the corresponding logic block (instance) required by the circuit. The optimization goals consist in placing connected logic blocks close together to minimize the required wiring (wire length-driven placement), and sometimes to place blocks to balance the wiring density across the FPGA (routability-driven placement) or to maximize circuit speed (timing-driven placement). The 3 major classes of placers in use today are min-cut (Partitioning-based) [2, 11], analytic [9, 14] which are often followed by local iterative improvement, and simulated annealing based placers [29, 24]. To investigate architectures fairly we must make sure that our CAD tools are attempting to use every FPGA's feature. This means that the optimization approach and goals of the placer may change from architecture to architecture. Partitioning and simulated annealing approaches are the most common and used in FPGA CAD tools. Thus we focus on both techniques in the sequel.

5) Routing:

Making sure that no routing resource is shared by more than one net is the goal of the FPGA routing challenge. The modern, state-of-the-art FPGA routing algorithm is path f inder [30]. A directed graph abstraction G(V, E) of the routing resources in an FPGA is used by Path F Inder. The IO terminals of logic blocks and the routing wires in the interconnect structure are represented by the set of vertices V in the graph. A potential link between two vertices is represented by an edge between them. A section of a routing graph in a mesh-based interconnect is shown in Figure 10. Finding a route for a certain network is the routing problem, is to identify a directed tree that connects each of the sink terminals of the net to the source terminal of G. Finding distinct, non-intersecting trees for every net in a netlist is a challenging task due to the finite number of routing resources in an FPGA. All of the nets in a netlist can be effectively routed by Path F Inder using an iterative, negotiation-based strategy. Nets are routed randomly without considering resource sharing during the initial routing iteration. The shortest path algorithm developed by Dijkstra [24] is used to route individual nets. Resources may be crowded at the end of the first iteration because several nets have used them. Based on the number of nets sharing a resource and the resource's history of congestion, the cost of utilizing that resource is increased over successive iterations. Nets are designed to bargain for routing resources as a result. Nets that can employ lower congestion alternatives are compelled to do so when a resource is extremely congested. On the other hand, a network may still use the resource if the alternatives are more congested than it is. Eq. below provides the cost of utilizing a routing resource n during a routing iteration.

$$cn = (bn + hn) \times pn.$$

bn is a function of the history of congestion during past iterations, pn is a function of the number of nets sharing the resource in the current iteration, and bn is the basic cost of utilizing the resource n. The hn term denotes the cost of using a resource that was shared during earlier routing iterations, while the pn term denotes the cost of using a resource that is shared. The latter term is based on the idea that a historically crowded node should seem pricey, even if it is only marginally shared right now. A detailed description of cost functions and a routing schedule may be found in [27].

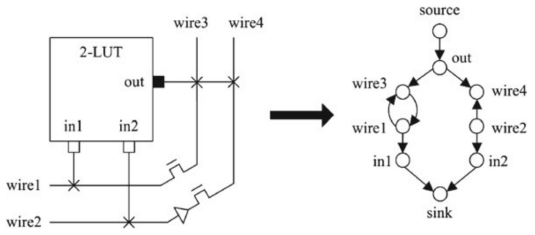

Fig. 9. FPGA routing architecture example

6) Timing Analysis:

Timing analysis [32] is used primarily for two tasks:

(1) assessing the speed of circuits that have been placed and routed in their entirety, and
(2) estimating the slack [31] of each source-sink connection during routing (placement and other stages of the CAD flow) in order to determine which connections should be made via fast paths to avoid slowing down the circuit.

The circuit under discussion is first shown as a directed graph. Input and output pins of circuit parts like LUTs, registers, and I/O pads are represented by nodes in the graph. The graph's edges represent the connections between these nodes. Between combinational logic Blocks' (LUTs') inputs and outputs, edges are inserted. The physical delay between the nodes is represented by a delay on these edges. The register input and output pins are not connected. A breadth first traverse of the graph is used to find the circuit's latency, starting at the sources (input pads, and register outputs). Next, using the following equation, the arrival times, Tarrival, at each node in the circuit are calculated:

Tarrival(i) = max j f anin(i)Tarrival(j) + delay(j,i)

If node I is the node for which calculations are currently being made and delay(j,i) is the edge's delay value from node j to node i. The circuit's delay is thus equal to the circuit's nodes' combined maximum arrival time, Dmax. Knowing how much delay may be added to a connection before the path that the connection is on becomes crucial might help direct a placement or routing algorithm. The slack of a connection refers to the maximum amount of delay that can be added to it before it becomes critical. One must determine the needed arrival time, Trequired, at each node in the circuit in order to calculate the slack of a link. First, we set the Trequired at all sinks (input registers and output pads) to Dmax. The required arrival time is then propagated backwards using the following equation, starting at the sinks:

Trequired (i) = min j f anout(i)Trequired (j) delay(j,i)

Finally, the slack of a connection (i, j) driving node, j, is defined as:

Slack(i, j) = Trequired (j) Tarrival(i) delay(i, j)

7) Bitstream Generation:

Bitstream data is created for a netlist after it has been put and routed on an FPGA. A bitstream loader is used to program this bitstream on the FPGA. Which SRAM bit of an FPGA should be set to 0 or 1 is indicated in the bitstream of a netlist. The technology mapping, packing, and placement data are read by the bitstream generator to program the SRAM bits of the look-up tables. The SRAM bits of connection boxes and switch boxes are correctly programmed using the routing details of a netlist.

VI. Summary and Conclusion

FPGA was introduced to the globe in 1985, and today it is crucial to the electronics industry. It easily lowers the NRE cost for the material and is mostly appropriate for prototype models. In some situations, FPGA is superior to ASIC. In this documentation, I provide a brief explanation of the existing application as well as the global routing architecture. It is anticipated that in the future, the ratio will climb more quickly. Finally, several strategies that have been used to lessen a few of FPGAs' and ASICs' drawbacks—either with or without sacrificing their main advantages—are explained.

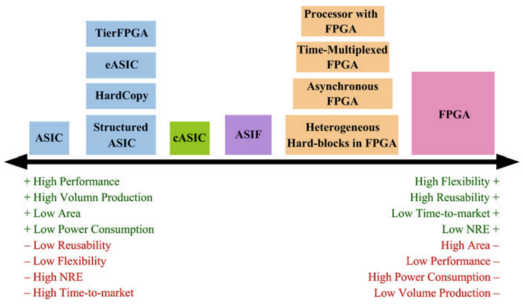

Fig. 10. Comparison of different solutions used to reduce ASIC and FPGA drawbacks

VII. Declaration of Originality

I, Evrard Leuteu Feukeu, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and

scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

REFERENCES

1. Actel: Proasic3 flash family FPGAs. http://www.actel.com/documents/pa3-ds.pdf (Oct 2005)
2. Dunlop, A., Kernighan, B.: A procedure for placement of standard-cell VLSI circuits. IEEE Trans. CAD, pp. 92–98 (Jan 1985)
3. Altera: http://www.altera.com (2010) 4. Altera: 40-nm FPGA power management and advantages. http://www.altera.com (2010)
5. Singh, A., Marek-Sadowska, M.: Efficient circuit clustering for area and power reduction in FPGAs. In: International Symposium on Field Programmable Gate Arrays, pp. 59–66 (2002)
6. Betz, V., Marquardt, A., Rose, J.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, New York (Jan 1999)
7. Brayton, R., Hachtel, G., Sangiovanni-Vincentelli, A.: Multilevel logic synthesis. Proc. IEEE 78(2), 264–300 (Feb 1990) 8. Brayton, R., McMullen, C.: The decomposition and factorization of Boolean expressions.Proc. ISCAS, 29–54 (1982)
9. Alpert, C.J., Chan, T., Huang, D., Kahng, A., Markov, I., Mulet, P., Yan, K.: Faster minimization of linear wirelength for global placement. In: ACM Symposium on Physical Design, pp. 4–11 (1997)
10. Compton, K., Hauck, S.: Automatic design of area-efficient configurable ASIC cores. IEEE Trans. Comput. 56(5), 662–672 (May 2007)
12. Huang, D., Kahng, A.: When clusters meet partitions: new density based methods for circuit decomposition. In: IEEE European Design and Test Conference, pp. 60–64 (1995) 11. Huang, D., Kahng, A.: Partitioning-based standard-cell global placement with an exact objective. In: ACM Symposium on Physical Design, pp. 18–25 (1997)
13. Bozorgzadeh, E., et al.: Routability-driven packing: metrics and algorithms for clusterbased FPGAs. IEEE J. Circuits Syst. Comput. 13(1), 77–100 (2004)
14. Sigl, G., Doll, K., Johannes, F.: Analytical placement: a linear or a quadratic objective function? In: Design Automation Conference, pp. 427–432 (1991) 15. Guterman, D.C., Rimawi, I.H., Chiu, T.L., Halvorson, R., McElroy, D.: An electrically alterable nonvolatile memory cell using a floating-gate structure. IEEE Trans. Electron Devices 26(4), 576–586 (April 1979)
16. Birkner, J., Chan, A., Chua, H.T., Chao, A., Gordon, K., Kleinman, B., Kolze, P., Wong, R.: A very-high-speed field programmable gate array using metal-to-metal antifuse programmable elements. Micro 23(7), 561–568 (Nov 1992)
17. Cong, J., Ding, Y.: Flowmap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA Designs. IEEE Trans. CAD, pp. 1–12 (1994)
18. Cong, J., Ding, Y.: On area/depth trade-off in LUT-based FPGA technology mapping. IEEE Trans. VLSI Syst. 2(2), 137–148 (1994) References 183
19. Cong, J., Hwang, Y.: Simultaneous depth and area minimization in LUT-based FPGA mapping. In: ACM/SIGDA International Symposium on Field Programmable Gate Array, pp. 68–74 (1995)
routing. In: ACM/IEEE Design Automation Conference, pp. 536–542 (1992) 20. Jia, X., Vemuri, R.: Studying a GALS FPGA Architecture Using a Parameterized Automatic Design Flow, pp. 688–693 (2006)
21. Hagen, L., Kahng, A.: Combining problem reduction and adaptive multi-start: a new technique for superior iterative partitioning. In: IEEE Trans. CAD, pp.92–98 (1997)
22. Dehkordi, M., Brown, S.: The effect of cluster packing and node duplication control in delay driven clustering. In: IEEE International Conference on Field Programmable Technology (ICFPT), pp. 227–233 (2002)
23. Murgai, R., Brayton, R., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/ area. In: IEEE International Conference on Computer Aided Design, pp. 6–9 (1991)
24. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press, Cambridge (1990) References 185
25. Carter, W., Duong, K., Freeman, R.H., Hsieh, H., Ja, Y.J, Mahoney, J.E., Ngo, L T., Sze, S.L.: A user programmable reconfiguration gate array. In: IEEE Custom Integrated Circuits Conference, pp. 233–235 (May 1986)
26. Lattice: Latticeecp/ec family data sheet versio 02.0. http://www.latticesemi.com/ lit/docs/datasheets/fpga/ecp$_e c_d atasheet.pdf (Sept 2005)$
27. Zakaria, H.: Asynchronous architecture for power efficiency and yield enhancement in the decananometric technologies: application to a multi-core system-on-chip. Ph.D. thesis, TIMA, Grenoble France (2011)
28. Murgai, R., Brayton, R., Sangiovanni-Vincentelli, A.: On clustering for minimum delay/ area. In: IEEE International Conference on Computer Aided Design, pp. 6–9 (1991)

29. Sechen, C., Sangiovanni-Vincentelli, A.: The timberwolf placement and routing package. JSSC, 510–522 (April 1985)

30. McMurchie, L., Ebeling, C.: Pathfinder: a negotiation-based performance-driven router for FPGAs. In: International Workshop on Field Programmable Gate Array, pp. 111–117 (1995)

31. Frankle, J.: Iterative and adaptive slack allocation for performance-driven layout and FPGA routing. In: ACM/IEEE Design Automation Conference, pp. 536–542 (1992)

32. Hitchcock, R., Smith, G., Cheng, D.: Timing analysis of computer-hardware. IBM J. Res. Dev. 100–105 (Jan 1983)

33. Ye, A.G., Rose, J.: Using bus-based connections to improve field-programmable gate-array density for implementing datapath circuits. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 14(5), 462–473 (May 2006)

34. Semiconductor, P.: Era60100 preliminary data sheet (1989)

35. El Gamal, A., Greene, J., Reyneri, J., Rogoyski, E., El-Ayat, K., Mohsen, A.: An architecture for electrically configurable gate arrays. IEEE J. Solid-State Circuits 24(2), 394–398 (April 1989)

36. Carter, W., Duong, K., Freeman, R., Sze, S.: A user programmable reconfiguration gate array. In: IEEE Custom Integrated Circuits Conference, pp. 233–235 (May 1986)

37. Wong, S., So, H., Ou, J., Costello, J.: A 5000-gate CMOS EPLD with multiple logic and interconnect arrays. In: Proceedings of the IEEE Custom Integrated Circuits Conference (1989), pp. 5–8 (2002)

38. DeHon, A.: Balancing interconnect and computation in a reconfigurable computing array (or, why you don't really want 100Field Programmable Array FPGA, Monterey, CA, pp. 69–78 (Feb 1999)

39. Betz, V., Marquardt, A., Rose, J.: Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, New York (Jan 1999)