More wires and fewer LUTs: A design methodology for FPGAs

Atsushi Takahara†, Toshiaki Miyazaki†, Takahiro Murooka†, Masaru Katayama†,
Kazuhiro Hayashi†, Akihiro Tsutsui†, Takaki Ichimori‡ and Ken-nosuke Fukami§
†NTT Optical Network Systems Laboratories
{taka, miyazaki, murooka, katy}@aecl.ntt.co.jp
{kazu, akihiro}@exa.onlab.ntt.co.jp
‡NTT Network Service Systems Laboratories
Ichimori.Takaki@nslab.ntt.co.jp
§NTT Science and Core Technology Laboratory Group
fukami@aecl.ntt.co.jp

Abstract

In designing FPGAs, it is important to achieve a good balance between the number of logic blocks, such as Look-Up Tables (LUTs), and wiring resources. It is difficult to find an optimal solution. In this paper, we present an FPGA design methodology to efficiently find well-balanced FPGA architectures. The method covers all aspects of FPGA development from the architecture-decision process to physical implementation. It has been used to develop a new FPGA that can implement circuits that are twice as large as those implementable with the previous version but with half the number of logic blocks. This indicates that the methodology is effective in developing well-balanced FPGAs.

1 Introduction

Today, FPGAs (Field Programmable Gate Arrays) are widely used in various applications. One of the advantages of using them is the short turnaround time for implementing given circuits compared to that with traditional gate arrays. However, the size of a circuit that can be implemented in an FPGA is limited. Although many types of FPGAs are commercially available now [10], a frequently asked question is: How many gates can be implemented in an FPGA? The upper limit for the number of logic blocks is specified in each FPGA manual, but this is often not enough information for users. For instance, when a circuit uses all the logic blocks in an FPGA, it is difficult to route all the nets in the circuit. Consequently, users find that they have to modify their circuits or buy more FPGAs to implement them. Moreover, even if the routing process is a success and 100% of the nets are routed, the operating speed of the implemented circuit may not meet the specifications.

The ideal FPGA would be one in which any number of logic blocks, within the upper limit, can be used to implement a circuit and the circuit meets the timing specifications. It is very difficult to design such an ideal FPGA for any type of circuit. Our first try led to the application-specific FPGA called *PROTEUS* [9]. The architecture of *PROTEUS* was determined by analyzing the logic functions used in telecommunications systems. In this analysis, the functionalities (counter, state machine, pattern matching) and the ratio of latches/Boolean gates in various telecommunications circuits were examined. The circuits implemented in *PROTEUS* can operate at the desired speed and they operate faster than circuits implemented in commercial FPGAs [13].

From our experience with *PROTEUS*, we became aware of some limitations in implementing circuits in the top-down manner, such as generating circuits for FPGAs from Register Transfer Level descriptions. *PROTEUS* has 2048 3-input Look-Up Tables (LUTs), but the average LUT utilization rate is around 20% using its dedicated CAD system. This means 80% of LUTs are not used efficiently, or in other words, that the architecture is extremely redundant for top-down design. We found that if some of these unused LUTs were replaced by wires and switches, routability was improved and more LUTs were usable. Essentially, this has become our design policy, i.e., "more wires and fewer LUTs". The current FPGA trend is to implement more LUTs than ever before[11, 3]. The wiring area should also be extended to avoid wasting the implemented LUTs. Therefore, we have to consider a balance between the number of LUTs and the number of wires and switches.

The question then is: How many wires and switches are adequate to improve the routability of a given number of LUTs? For FPGAs, we need to consider not only the architecture but also the CAD system used to implement application circuits. In designing *PROTEUS*, we could not determine the effects of the CAD system because we developed the CAD system in parallel. This caused disadvantages in top-down design. The methods of evaluation are important in finding a good balance among wires, switches and LUTs because there are so many parameters involved and the evaluation of a single portion of these parameters only leads to a local optimal solution.

In this paper, we present an FPGA design methodology that covers all aspects of FPGA development from the architecture-decision process to implementation techniques, considering the enhancement of the routability of the *PROTEUS* FPGA chip. First, several architecture candidates are evaluated using the FPGA architecture and CAD system testing environment (*FACT* [7]). FACT is similar to the evaluation tool proposed by Betz et al.[1]. They concentrated on the number of tracks considering the shapes of an FPGA. In our evaluation, the specific switch patterns are also considered. The routability of circuits is examined by changing the number of wires and switches and the final architecture is determined based on this examination.

We then consider the physical implementation aspects, i.e., the chip size and the propagation delay. Since we increase the number of wires and switches, the layout structure of a chip is much more complicated. We modify the switch structures and the number of LUTs taking physical implementation into consideration.

Based on the above considerations, we developed a new FPGA chip (*PROTEUS-Lite*)[6] using $0.5\mu m$ CMOS process technology with three metal layers. Since there are innumerable wires and switches, we prepared design tools to handle the relationship between these complicated switch structures and their control memories. Moreover, FPGA testing is very difficult. Although sophisticated method for testing LUTs has been proposed [4, 12], testing methods for wires and switches have not yet been discussed in depth. We developed a semi-automatic test pattern generator for wires and switches that generates 100% coverage test patterns using the structural information of an FPGA. The results of our evaluation indicated that our new FPGA achieves near 100 % routability within an 80% LUT utilization rate in our top-down CAD systems. This confirms that our design policy of "more wires and fewer LUTs" is effective.

The rest of this paper is organized as follows: Section 2 explains our architecture evaluation method. Section 3 discusses the physical considerations of chip size and propagation delay. Section 4 presents the *PROTEUS-Lite* FPGA, its specifications, and our FPGA-specific design tools, i.e., the verification tool and the semi-automatic test pattern generation tool. Section 5 evaluates our new FPGA architecture with practical telecommunications circuits. Section 6 concludes the paper.

2 Architecture Evaluation

An FPGA consists of logic blocks for logic functions and a wiring area containing wires and switches that connect the logic blocks. Here, we assume that an FPGA consists of the set of logic blocks, surrounding wires and switches shown in Fig. 1. Our logic block consists of four 3-input LUTs, a 5-

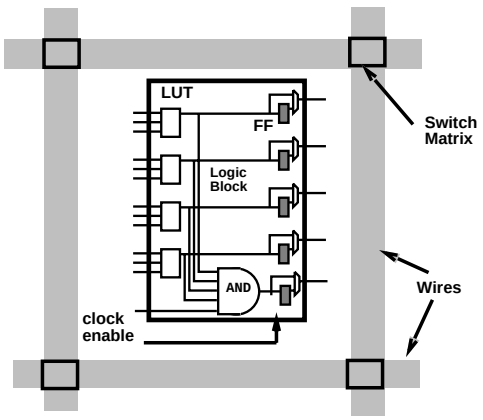


Figure 1: An FPGA structure

input AND gate, five FFs, and selectors. Each block has 14 inputs and five outputs in total. This is the same structure as for *PROTEUS*.

2.1 Wiring Structure Candidates

The routability of an FPGA depends on the number of wires and switches. We evaluated the wiring structure with respect to the following three points.

1. The I/O bandwidth of a logic block. This parameter indicates how many alternative connections are possible for the inputs and outputs of a logic block.
2. The number of wires.
3. The number of switches.

The I/O bandwidth of a logic block represents the possibility of connecting one logic block to another. If inputs have connections in two different directions, for instance, north and south, this extends routing possibilities. Since the wires are also used to connect other logic blocks that are not adjacent to them, there are more horizontal/vertical wires than the number of logic block inputs and outputs. Regarding the number of switches, the maximum number is proportional to the number of wires and the I/O bandwidths of a logic block. This number would be very huge in the case where any connection between wires is possible. Consequently, the number of switches should be reduced and the effect of this reduction on routability should be minimized.

In this paper, we consider the three types of structures shown in Fig. 2. Type A was prepared in order to evaluate

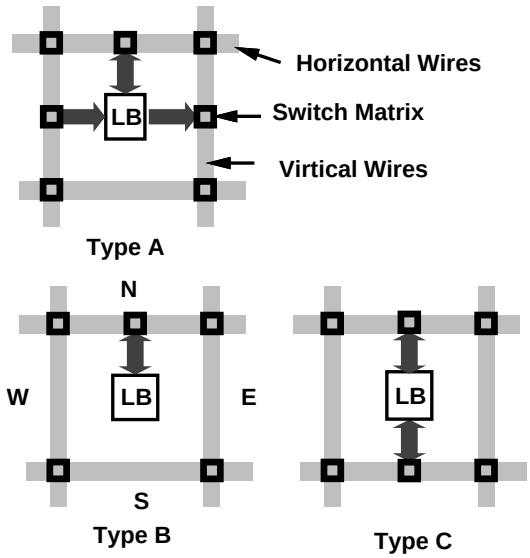


Figure 2: Switch pattern types

the effect of the number of switches without changing the number of wires. In types B and C, the I/O bandwidths of the logic blocks are different. In both, horizontal lines are used for the input/output connections of logic blocks and vertical lines are used for the connecting wires.

For each type, we evaluate several structures which have the different numbers of lines and switches. The specifications for these structures are in Table 1.

2.2 Co-Evaluation

The candidate structures are evaluated using the *FACT* system. Each candidate structure is described in an Architec-

Table 1: Design candidates

Type	No. of lines		I/O Direction				No. of
	H	V	N	W	S	E	switches
A	20	20	19	14	0	5	374–1170
B	30–32	8–16	19	0	0	0	384–797
C	30–32	10–16	19	0	19	0	399–410

(H:horizontal, V:vertical)

ture Definition Format (*ADF*) from which *FACT* can produce the evaluation environment. Using this environment, we estimated the routability of each structure using practical circuits.

The results for type A candidate structures are in Table 2. In the experiments, the number of switches was varied. In

Table 2: The effect of the number of switches

structure	N-W	W-LB	N-LB	E-LB	Total	U.N.
v6	40	90	170	80	410	111
v8	40	100	150	50	390	55
v10	58	133	100	33	374	47
v12	400	380	280	100	1170	0

U.N.:unrouted Nets

the table, "v12" is a rich switch pattern in which there are switches on all points where lines cross. This is the upper limit for the number of switches. In the other structures, we reduce the number of switches. Table 2 suggests the possibility of finding a "fewer-switches" pattern that preserves efficient routability.

The routability of type B structures is in Fig. 3. Here, the number of horizontal/vertical wires is varied. In type

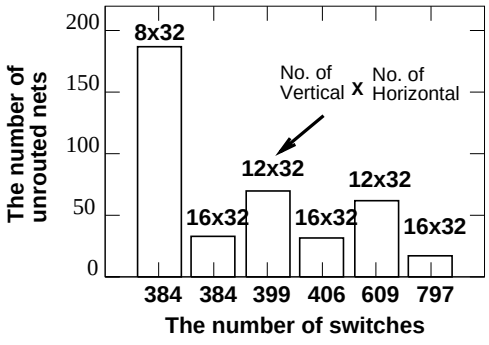


Figure 3: The effect of the number of switches and the number of wires in type B

B structures, the inputs and outputs of a logic block are connected to horizontal lines on the north side. The number of vertical wires also affected the final routing results. These wires provide effective alternate routes for the routing tools to improve the routability of nets, but this routability is not greatly improved compared to the optimal circuits of type A. Due to the limited I/O bandwidth, the routing tool uses the same horizontal lines. This causes congestion in the routing resources, which degrades routability.

In type C, the I/O of a logic block is connected to north-side horizontal lines as well as south-side horizontal lines. This makes the routing path for the inputs and outputs of a logic block richer than for type B. The effect of the number of wires was examined with the switch pattern fixed. The results are in Fig. 4. Routability largely depends on the

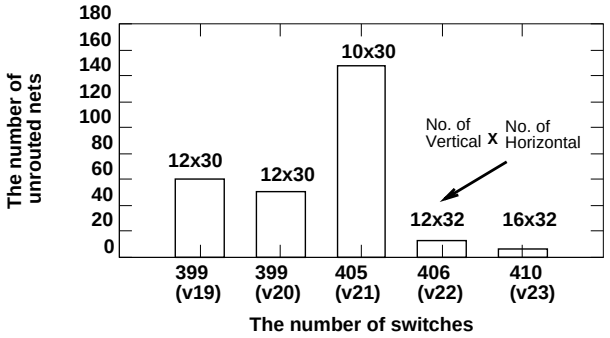


Figure 4: The effect of the number of switches and the number of wires in type C

number of wires. "v22" and "v23" are optimal for all candidate structures. The number of switches each in "v22"and "v23" is almost four times that in *PROTEUS*, which has 109 switches with 13 vertical lines and five horizontal lines. Using the *PROTEUS* structure, 875 wires are not routed, so the routability of our optimal candidates is much improved.

From the above evaluation, we can obtain better routability in both type B and type C when the ratio between the number of horizontal wires and vertical wires is 2:1. This finding is the same as that of [1]. On the number of switches, type C achieves good routability with fewer switches than the other types.

3 Physical Implementation Aspects

The physical implementation aspects have not been considered in previous discussions. Increasing the number of switches and wires affects chip size and the propagation delay of signals.

3.1 Area

Since the maximum chip size is limited, it is inevitable that the number of logic blocks will have to be decreased when more wires and switches are implemented to enhance routability. This limitation underlies our design policy, which is to increase the number of wires and decrease the number of logic blocks.

Here, we present an area comparison of our two FPGA designs to show how the area increases when more wires and switches are implemented. Table 3 is an area comparison between *PROTEUS* and *PROTEUS-Lite* chips whose wire structure is the same as "v23" in Section 2. The numbers for Switches, Wires and Memories are the numbers of each item of the Basic Element (BE), which is a repeated layout block in each chip. Here, Memories include LUT memories and control memories for switches. The Area is the ratio of the two chips. The area of *PROTEUS-Lite* is twice that of *PROTEUS*.

Table 3: Area comparison between *PROTEUS* and *PROTEUS-Lite*

	PROTEUS	*PROTEUS-Lite*	Ratio
Switches	185	425	2.29
Wires	37	63	1.70
Memories	256	425	2.53
Area	1	2.12	2.12

The area-increase ratio is slightly lower than that of switches and memories, but it is higher than that of wires. We compared the number of wires visible to users in each chip. In an actual layout, additional wires connecting wires and switches, or switches and memories are required. The area taken up by these wires should be considered in the area penalty. This is an example of implementation; different implementations have a different ratio. Currently, we use three metal layers. The area penalty may be decreased by using more metal layers. At this time, we paid a double penalty for the area.

3.2 Delay

Another important aspect is the signal propagation delay of wires and switches. Signal propagation delay depends on the number of switches that the signal passes through [2]. This is partially true. The unused switches or off-switches affect the propagation delay as well.

To avoid increases in propagation delay caused by unused switches, we installed a buffered switch consisting of two tristate drivers between the cascading switch matrixes. This localizes the load and limits the propagation delay.

Consider the propagation delay of the switch matrix shown in Fig. 5. There are 16 × 64 normal switches, which are divided by buffered switches. Figure 6 shows the propagation delay through 64 normal switches and buffered switches calculated by SPICE simulation, where the number of buffered switches is varied from 0 to 15. In this example, if the number of buffered switches is less than 4, the propagation delay decreases because the buffered switches can effectively drive the signal against the load of the unused switch. On the other hand, if the number of buffered switches is more than 4, the propagation delay increases because the intrinsic delay of the buffered switches becomes dominant. Here, we exploit the ratio between the number of buffered switches and normal switches. Consequently, a buffered switch was inserted at every 16 vertical lines of the circuit depicted in Fig. 5 to minimize the propagation delay.

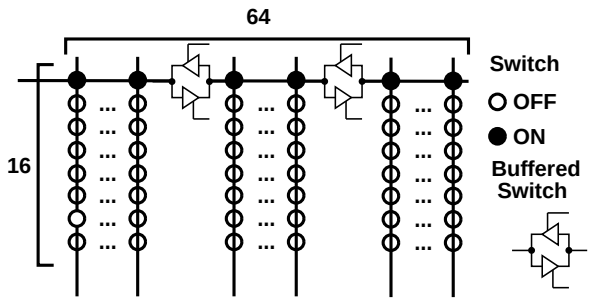


Figure 5: Sample circuit using buffered switches

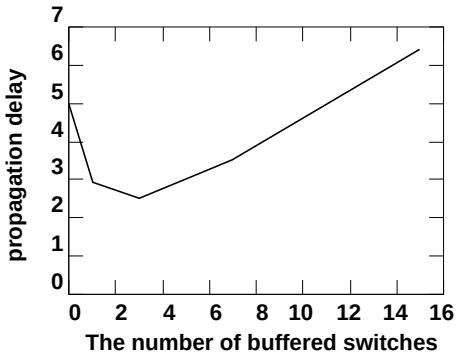


Figure 6: The effect of buffered switches

The buffered switches have another function: They divide a wire into several segmented wires. This provides more alternative routes.

4 Implementation

We designed the new FPGA chip, *PROTEUS-Lite*, based on our design policy after doing the evaluations previously explained. This section describes the specifications for *PROTEUS-Lite* and our design flow.

4.1 PROTEUS-Lite Chip

A unique feature of the *PROTEUS-Lite* chip is that the wiring area is much larger than the logic area. Figure 7 shows the logical structure of the Basic Elements (BEs) of the *PROTEUS-Lite* chip. The inputs and outputs of a logic block are connected to 32 upper/lower middle wires. These wires can be separated by buffered switches, which are located between the input switch matrix and the output switch matrix, because logic block inputs and outputs are usually connected to different wires.

Two other kinds of wires, long ones and local ones, were also prepared. Horizontal/vertical wires whose lengths are half the chip are mainly used to connect logic blocks to primary I/Os. The local wires are used to connect adjacent logic blocks. Table 4 shows the statistical data for the switches, memories and wires in a BE.

Table 4: Statistical data for switches, memories and wires

Switch	Memory	Wires				
		Long		Middle		Local
		H	V	H	V	
425	648 bit	6	4	32	16	5

H:horizontal, V:vertical

The *PROTEUS-Lite* chip consists of 28 × 10 BEs. Figure 8 is a photograph of the chip along with its specifications which are compared to those of the *PROTEUS* chip.

4.2 Design Verification Flow

PROTEUS-Lite is a full custom chip, and it was designed manually. Verification of the design is critical due to its com-

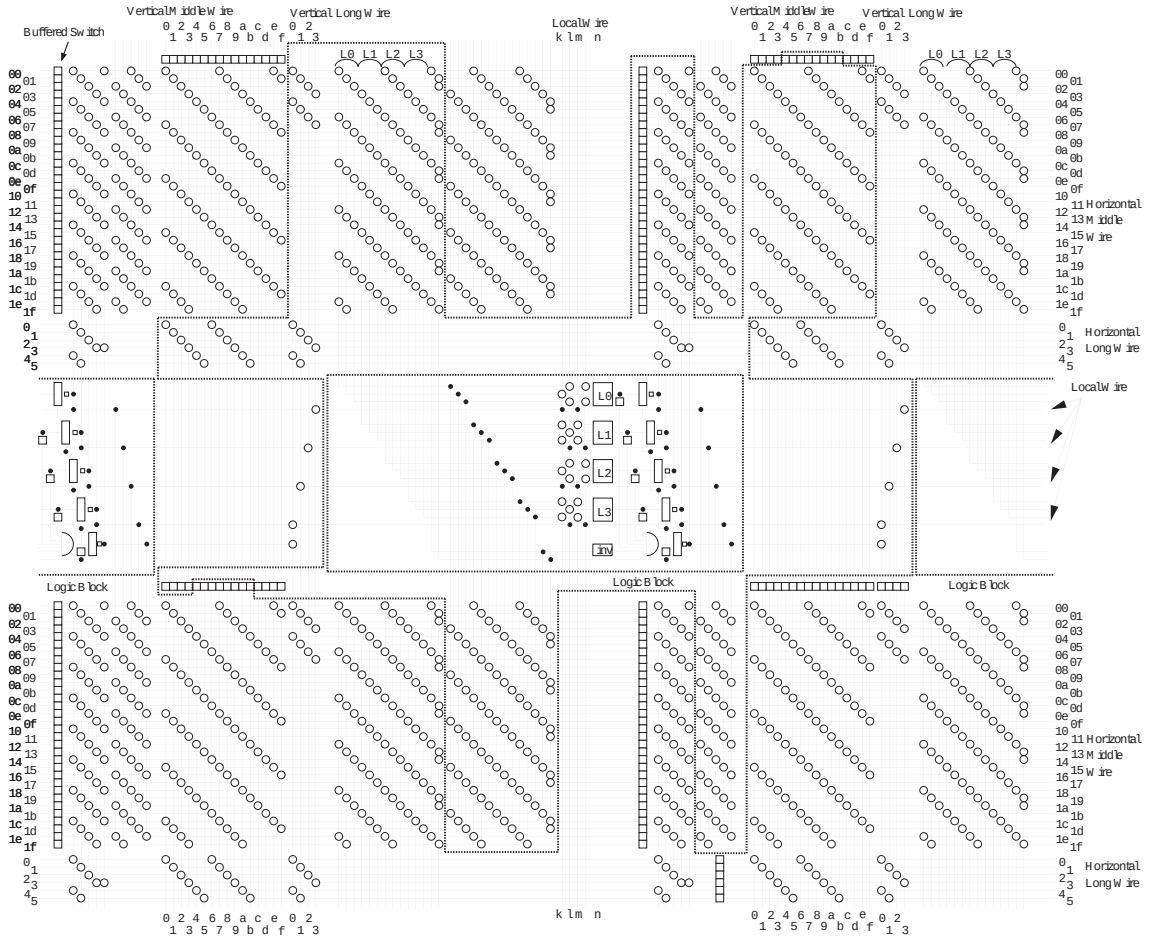
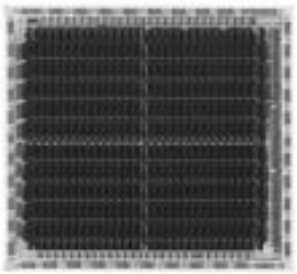


Figure 7: *PROTEUS-Lite* switch pattern

Die Photograph



Specifications

	PROTEUS-Lite	PROTEUS
FPGA Type	SRAM type FPGA	SRAM type FPGA
Process	0.5μCMOS(3 metals)	0.5μCMOS(3 metals)
Chip size	17.1mm×15.3mm	15.0mm×15.0mm
I/O	136 (bi-directional)	192 (bi-directional)
Logic Blocks	280 (28×10)	512 (32×16)
Flip Flops	1672	2722
Configuration memory	223K bit	140K bit
Packages	QFP/CSP/PGA	PGA

Figure 8: *PROTEUS-Lite* chip

plicated switch patterns. Moreover, it is hard to provide the correct specifications for the layout design. To overcome these difficulties, we developed design tools to generate the correct specifications. Our design method uses the *FACT* system for evaluation. In this system, the FPGA architecture is formally defined in the Architecture Definition Format (*ADF*), in which the logical structure of the switches, wires and their connections as well as the structure of the logic blocks are described. In the evaluation phase, the *ADF* is verified by applying many sample circuits. Consequently, the correct specifications for the *PROTEUS-Lite* chip can be obtained from the *ADF*.

Our design verification flow is shown in Fig. 9. Netlist information on *PROTEUS-Lite* is generated from the *ADF* as well as the library, in which LUTs, FFs, and switches are defined. In the *ADF*, there is no information on the configuration of memory structures, so these structures are defined in the memory definition file (*MDF*). In the *MDF*, the relationship between each switch and its memory addresses are defined. The *PROTEUS-Lite* design was verified by comparing the netlist extracted from the layout patterns with the netlists generated from the *ADF*, *MDF*, and library.

4.3 Semi-automatic Test Pattern Generation

Another important aspect of designing FPGAs is how to test the memories, logic blocks, wires and switches. The memories can be tested using the standard memory testing method, and there are several techniques to test the logic blocks. However, it is difficult to develop a general testing method for the wires and switches because the test method largely depends on their structures. Our approach is to test semi-automatically using the information in the *ADF*.

Our testing method is explained here using the 3 × 3 switch matrices in Fig. 10. The fault patterns of wires and switches are categorized into 1) stuck-at faults in switches and 2) bridge/open faults in wires. These faults are tested by feeding the patterns that satisfy the following conditions.

Condition 1 Any input to the switch matrix is fed by patterns 0 and 1.

Condition 2 Any input to the switch matrix is fed by patterns in which there are signal transitions of $1 \rightarrow 0$ and $0 \rightarrow 1$.

Condition 3 Each input signal pair (a, b) is fed by two patterns $(a, b) = \{(1, 0), (0, 1)\}$.

The above patterns can be obtained by generating m-out-of-n codes, where m is the number of inputs to a switch matrix and n is equal to $m/2$. The patterns in which $m/2$ 0s and 1s are continuous are not used because both signal transitions ($0 \rightarrow 1$, $1 \rightarrow 0$) are not included in these patterns.

To test switches and wires, the specific circuits in which the test patterns are fed to the switch/wire under test should be configured in the FPGA. Each switch should be set to on and off. For efficient testing, the circuit should cover as many switches as possible, but only one of switches in horizontal and vertical wires can be tested at a time. Therefore, we have to find patterns which satisfy this condition. For the case in Fig. 10, four circuits are required in order to test 9 switches/wires in this switch matrix.

Our developed tool generates the circuits for testing switches and wires from both the *ADF* and a typical switch matrix pattern library in which the templates for position independent switch patterns are defined. This tool finds the specific switch patterns that match one of the templates in the library and generates the circuits for testing switches in the target switch matrix. 25130 circuits are generated by our tool to test 100% of the switches in *PROTEUS-Lite*. A chip is tested by repeating the configuration for each circuit and feeding the test pattern. Since the number of circuits is large, we use the Direct Memory Programming Mode, which can configure arbitrary bits of the configuration memory. Each circuit is configured by only writing different bits of the configuration memory from that of the previous circuit for efficient testing.

5 Evaluations and discussions

We evaluated *PROTEUS-Lite* by comparing it to *PROTEUS*, which has more LUTs but fewer wires than *PROTEUS-Lite*. In this evaluation, we used practical telecommunications circuits designed to handle ATM protocols. The routability of each chip is shown in Fig. 11, where the X-axis is the number of used LUTs compared to that of LUTs implemented in a chip and the Y-axis is the routability ratio between routed nets and all nets in each circuit. These results were generated using each dedicated CAD tool. The current *PROTEUS-Lite* CAD system is an extension of our original *PROTEUS* CAD system [13]. Our placement tool uses the simulated annealing method which considers the distance between LUTs and directions of signals in its evaluation function. Our routing tool consists of several routing procedures that are based on the line-search method or the Dijkstra method. Each procedure uses architecture-dependent heuristics to achieve both high-quality routing results and a short processing time[8].

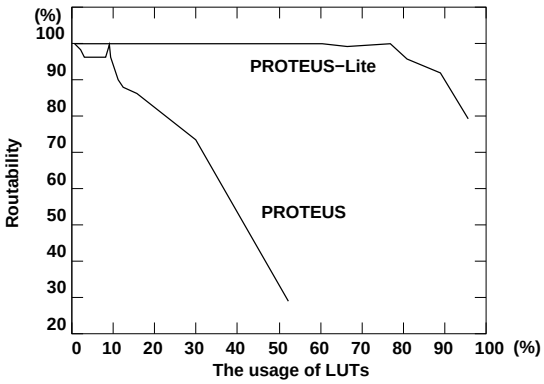


Figure 11: Routability comparison

From the results in Fig. 11, nets of circuits which use up to 80% of the LUTs are almost completely routed by the *PROTEUS-Lite* dedicated CAD system. In *PROTEUS*, nets in circuits which use up to 20% of the LUTs are routed by its dedicated CAD system. Let us consider these results from the view point of wasted LUTs. There are 2048 LUTs in *PROTEUS* and 1120 LUTs in *PROTEUS-Lite*. Let us calculate the number of realistically usable LUTs according to the above results. *PROTEUS*' realistic LUT usability is

$$2048 \times 20\% = 409.6$$

and *PROTEUS-Lite*'s realistic LUT usability is

$$1120 \times 80\% = 896.$$

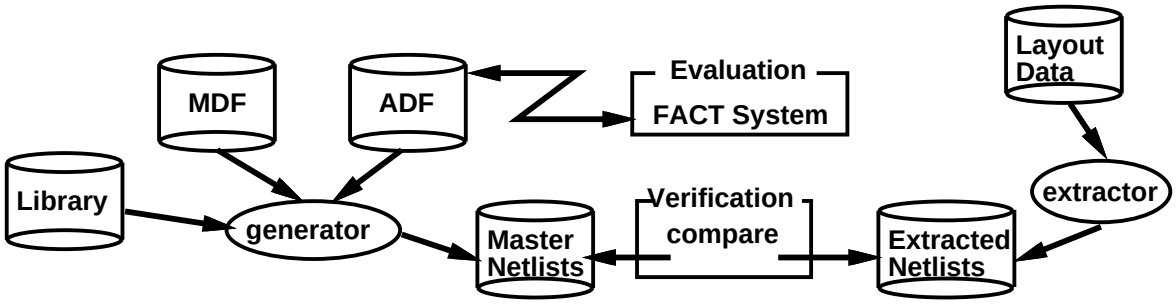


Figure 9: Design verification flow of *PROTEUS-Lite*

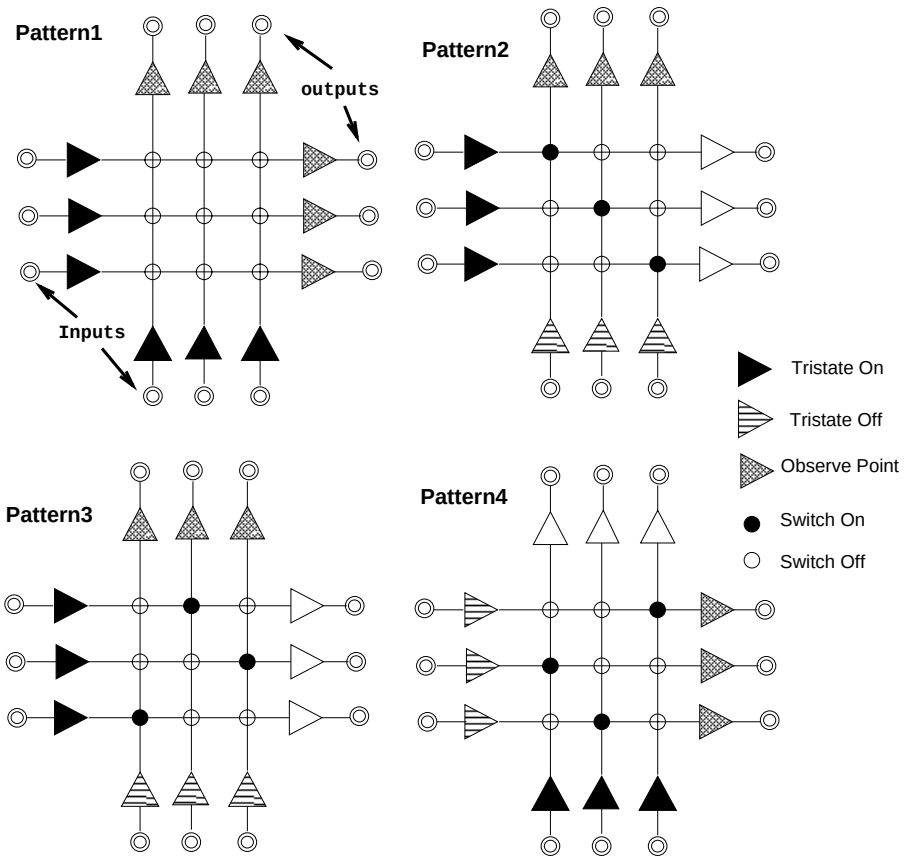


Figure 10: Configuration patterns for testing a switch matrix

The number of usable *PROTEUS-Lite* LUTs is nearly twice the number of usable *PROTEUS* LUTs, while the total number of LUTs implemented in *PROTEUS-Lite* is half that in *PROTEUS*. A chip with fewer LUTs can realize larger circuits. This confirms the validity of our design policy of "more wires and fewer LUTs".

We compared *PROTEUS-Lite* to a commercial FPGA. The gate capacity of *PROTEUS-Lite* is about 19.8K gates using Xilinx's calculation method where a 3-input LUT is 3.2 gates, a 4-input LUT is 7 gates and an FF is 10 gates. XC4010E, which has 18.1K gates, is similar to *PROTEUS-Lite*, while *PROTEUS* is similar to XC4013E, which has 25.3K gates. Table 5 shows the critical path delay of several telecom circuits which are implemented in *PROTEUS-Lite* and XC4010E-5[1]. These results were obtained using their specific CAD tools. The implemented circuits in *PROTEUS-Lite* are faster or equal to those in XC4010E-5. The delay calculated by our method is accurate compared to the actual delay [5] while Xilinx's delay calculation tool (xdelay) sometimes estimates critical path delay that is shorter than the actual delay, from our experience. *PROTEUS-Lite* has numerous wires so that placement affects critical path delay. We are developing a new placement tool that will take the application circuit structure into consideration. This tool will make critical path delay shorter using the advantages of *PROTEUS-Lite*'s structure.

Table 5: A comparison of critical path delay

Circuit	*PROTEUS-Lite*			*XC4010E-5*[2]	
	# of LUTs	# of FFs	Delay (ns)	# of LUTs	Delay (ns)
fcount	45	13	31.7	31	36.8
rts_mng	46	17	27.2	21	54.9
v_atgp	49	15	39.3	29	37.3
s_vtg	50	13	33.5	32	36.0
sc_mng	70	11	40.1	39	40.1
au4gen	75	21	41.4	51	48.5
hec	85	48	22.3	63	22.6
vc4gen	89	13	59.5	56	59.3
cellgen	136	54	53.8	98	55.1
aall_t	214	59	51.1	125	59.5
stm1gen	228	93	76.2	159	75.3

6 Conclusions

We presented a design methodology for FPGAs that covers their development from the architecture-evaluation process to implementation. The underlying policy of the methodology is "more wires and fewer LUTs". *PROTEUS-Lite*, which was designed accordingly, demonstrated an optimal solution for FPGA architecture that is usable for top-down design methods. The total number of LUTs is half that in our previous chip, but there is twice the number of usable LUTs. This confirms that our design methodology is effective and it shows that the balance between logic blocks and wiring resources is very important in designing FPGAs.

Acknowledgments

The authors would like to thank Mr. Takumi Watanabe and Mr. Kenji Ogura for their useful comments on the physical layout of the chip. They also wish to thank Dr. Naohisa Ohta, Mr. Tadanobu Nikaido, Dr. Junji Suzuki and Mr. Kazuyoshi Matsuhiro for discussions and encouragement.

[1] The speed grade is 5.

[2] The number of FFs is the same as that in *PROTEUS-Lite*.

References

[1] Betz, V., and Rose, J. Directional Bias and Non-Uniformity in FPGA Global Routing Architectures. *Proc. ICCD'96* (1996), 652–659.

[2] Brown, S., Khellah, M., and Vranesic, Z. Minimizing FPGA Interconnect Delays. *IEEE Design and Test of computers Winter* (1996), 16–23.

[3] Bursky, D. Programmable Arrays Mix FPGA and ASIC Blocks. *Electronic Design* (October 1996), 69–74.

[4] Hermann, M., and Hoffmann, W. Fault Modeling and Test Generation for FPGAs. *Proc. 4th International Workshop on Field-Programmable Logic and Applications, FPL '94* (1994), 1–10.

[5] Katayama, M., Takahara, A., Miyazaki, T., and Fukami, K. Delay Calculation Method for SRAM-based FPGAs. *IEICE Trans. Fundamentals E80-A*, 9 (1997), 60–65.

[6] Miyazaki, T., Takahara, A., Katayama, M., Murooka, T., Ichimori, T., Fukami, K., Tsutsui, A., and Hayashi, K. CAD-oriented FPGA and Dedicated CAD System for Telecommunications. *Proc. 7th International Workshop on Field-Programmable Logic and Applications, FPL'97 (LNCS 1304)* (1997), 11–20.

[7] Miyazaki, T., Tsutsui, A., Ishii, K., and Ohta, N. FACT: Co-evaluation Environment for FPGA Architecture and CAD System. *Proc. 6th International Workshop on Field-Programmable Logic and Applications, FPL'96* (1996), 34–43.

[8] Murooka, T., Takahara, A., Miyazaki, T., and Tsutsui, A. An architecture-oriented routing method for FPGAs having rich hierarchical routing resources. *Proc. ASP-DAC'98* (1998), to be published.

[9] Ohta, N., Nakada, H., Tsutsui, A., and Miyazaki, T. PROTEUS: Programmable Hardware for Telecommunication Systems. *Proc. ICCD'94* (1994), 178–183.

[10] OptiMagic. *Programmable Logic Jump Station, http://www.optimagic.com/.*

[11] Reddy, S., Cliff, R., Jefferson, D., Lane, C., Sung, C. K., Wang, B., Huang, J., Cope, T., McClintock, C., Leong, W., Ahanin, B., and Turner, J. High Density Embedded Array Programmable Logic Architecture. *Proc. IEEE Custom Integrated Circuits Conference* (1996), 251–254.

[12] Stroud, C., Lee, E., Konala, S., and Abramovici, M. Using ILA Testing for BIST in FPGAs. *Proc. IEEE International Test Conference* (1996), 68–75.

[13] Tsutsui, A., and Miyazaki, T. An Efficient Design Environment and Algorithms for Transport Processing FPGA. *Proc. ASP-DAC'95/CHDL'95/VLSI'95* (1995), 791–798.