Journal of Universal Computer Science, vol. 6, no. 2 (2000), 227-239
submitted: 8/7/99, accepted: 29/11/99, appeared: 28/2/00 © Springer Pub. Co.

The Price of Routing in FPGAs

Florent de Dinechin
Projet Arénaire, LIP-CNRS-INRIA,
École Normale Supérieure de Lyon, France
Florent.de.Dinechin@ens-lyon.fr

Abstract: Studying the architectural evolution of mainstream field programmable gate arrays (FPGAs) leads to the following remark: in these circuits, the proportion of silicon devoted to reconfigurable routing is increasing, reducing the proportion of silicon available for computation resources. A quantitative analysis shows that this trend, if pursued, will lead to a widening gap between FPGA performance and VLSI performance. Some prospective solutions to this problem are discussed.
Key Words: FPGA, reconfigurable computing, routing resources, hardware complexity
Category: B.7

1 Introduction

1.1 The FPGA success story

Any hardware data-processing application requires some kind of dedicated logic circuitry. As a minimum, when using only "off-the-shelf" components, the designer needs some amount of "glue logic" to interface these components together, and to interface them with the rest of the world (be it a computer backplane). But dedicated hardware may also be needed for the bulk of the data processing itself, when no off-the shelf component is available or when their performance is inadequate.

Until recently, to implement such dedicated logic, the designer had to choose among the following options:

- using discrete components, which takes up a lot of physical space and is very inefficient, and is therefore only suited for very small amounts of glue logic,
- using some off-the-shelf processor or micro-controller, which is very flexible due to programmability, but slow and poor in inputs/outputs,
- designing an application-specific integrated circuit (ASIC), which guarantees the best results, but needs several months and is expensive unless in the context of high-volume production.

Field programmable gate arrays (FPGAs) were therefore designed as off-the-shelf VLSI circuits able to emulate arbitrary logic, with performances close to those of an ASIC, but with the flexibility of software. Typically, a designer evaluates the requirements of his dedicated circuitry (number of gates, number of inputs and outputs, speed), buys an FPGA matching these requirements, and programs it to get an ASIC replacement. The tools used to program FPGAs are very similar to those used to design ASICs (for example they input standard hardware description languages such as VHDL or Verilog), but the designer may test his circuit instantly, instead of having to go through the lengthly and costly ASIC foundry process. Besides, some FPGAs are reconfigurable, which allows for hardware debugging and upgrading. Being an off-the-shelf component produced in high volume, the cost of an FPGA in development time is a fraction of the cost of an ASIC.

FPGAs have therefore been among the fastest growing sectors of the silicon industry in the last years. They are mostly used as glorified glue logic, replacing ASICs where time to market is critical, or where a small production volume wouldn't justify the cost of an ASIC. Moreover, they have also spawned new applications and research interests, from rapid prototyping of VLSI [SKC+95] to general-purpose hardware acceleration for numerical applications [Vui94, DeH94, Bre98].

1.2 FPGA architectures

The FPGA architecture that allows to achieve these goals typically consists of a large number of configurable logic blocs embedded in a network of configurable interconnections [RM98]. We are not concerned here by the exact nature of the logic blocs (which also contain some memory elements), neither will we discuss the topology of the network, although these two questions have been and still are the subject of abundant research and discussion. We will not escape such discussion later in section 4, but until then an abstracted view of typical FPGA architectures, such as depicted on Fig. 1, will suffice.

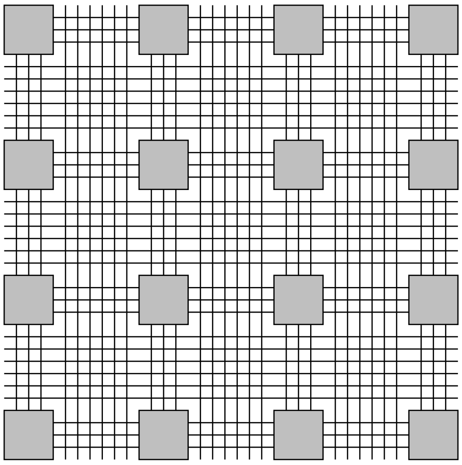

Figure 1: A typical FPGA architecture. Grey boxes are functional units, and lines are wires. To ensure that the interconnect network is programmable, there must be some sort of programmable switch (not shown) wherever two lines cross.

What Fig. 1 doesn't show is that the *programmability* of the FPGAs has a significant hardware cost: there must be switches on the wires to make the interconnect network configurable, and there must be some kind of memory holding this configuration along with the configuration of the logic blocks. Finally there must be some dedicated logic and routing to allow for the loading of this configuration.

All this consumes a significant proportion of the silicon of a chip, with implications on both capacity and speed of the FPGA. Thus, at a given time, an FPGA built using a state-of-the-art VLSI process cannot hold the biggest full-custom circuits which can be fabricated in the same process. Moreover, for those circuits which are small enough to fit in the FPGA, this FPGA implementation is slower than its full-custom counterpart.

This cost may be expressed as a *time lag* between FPGA and full-custom. For example, in 1999, one may observe that top-of-the-range FPGAs from various vendors claim to contain the equivalent of one to two million programmable gates, with system-level operating frequencies above 100MHz. These numbers are comparable with the transistor count (3.1 million, meaning roughly 1 million gates) and the operating frequency (60 to 120 MHz) of Pentium processors built in 1993-1994 : here we have a time lag of five or six years.

One may think that this time lag is constant, which would mean that we only need to wait for another five years to get the current full-custom performance out of an FPGA. The purpose of this paper is to question this assumption: we will show that the current trend in FPGA technology, if pursued, will entail an increase in this time lag. The reason for it is that an ever increasing *proportion* of FPGA silicon is devoted to the routing architecture, and therefore wasted for computing itself. Therefore the curve of FPGA performance doesn't follow the curve of VLSI integration.

1.3 Rent's rule and FPGAs

Table 1 gives a summary of the evolution of routing resources per look-up table (or LUT, the computing unit) in the three most recent FPGA families from Xilinx [Xil97, Xil98]. Figure 2 shows a logical view of an actual recent FPGA, the Xilinx Virtex [Xil98].

This table shows that the amount of routing resources per LUT increases almost proportionally to the number of LUTs per line or column. This general trend [TMM+98] may be explained by the FPGA version of Rent's experimental law [SKC+95, Ull84] which states that, as a logic circuit is partitioned, the number of signals crossing the boundary of the partition is proportional to the number of gates on each side, raised to a power r which ranges from 0.5 to 0.8, depending on the class of application.

Let us transpose this rule to our FPGA (an extensive review on this kind of problems in the context of FPGAs may be found in section 7.6 of DeHon's thesis [DeH96]). Let us consider a vertical line splitting an FPGA in two halves. If N is the number of LUTs in a row/column, then there are $O(N^2)$ gates on each side of the partition. Rent's rule tells that the FPGA needs $O(N^{2r})$ wires across the vertical line. This line crosses N channels, so we need $O(N^{2r-1})$ wires per channel. For r between 0.5 and 0.8, the exponent $2r-1$ ranges from 0 (constant channel width) to 0.6.

However, the value of $2r-1$ extracted from Table 1 seems to be quite close

Series	**LUT matrix**[1]		**Routing wires per LUT** (horiz.+vert.)
	min	max	
4000E	$10 \times 10 \times 2 = 200 \approx \mathbf{14^2}$	$32 \times 32 \times 2 = 2048 \approx \mathbf{45^2}$	21
4000X	$14 \times 14 \times 2 = 392 \approx \mathbf{20^2}$	$56 \times 56 \times 2 = 6272 \approx \mathbf{79^2}$	38.5
Virtex	$16 \times 24 \times 4 = 1536 \approx \mathbf{39^2}$	$64 \times 96 \times 4 = 24576 \approx \mathbf{157^2}$	67.5

Table 1: Xilinx FPGAs

to 1. This can be explained in several ways:

- FPGAs are still in their infancy, and extrapolating asymptotic laws from Table 1 is obviously hazardous.
- An FPGA has to accomodate any application, and thus implements a very conservative, worst case Rent's exponent.
- In the FPGA market, the speed of the design cycle is an increasingly important selling point. The longest step of this design cycle is the (NP-complete) place-and-route phase, and with current methodologies and heuristics, this step greatly benefits from an excess in routing resources. The lost silicon area is then more than compensated by the improvement in FPGA design time [RH97].

Whatever the exact growth of routing resources, one obvious consequence is that the *proportion* of silicon devoted to routing increases with integration. This means in turn that routing has an increasing impact on FPGA performances. This is all the more true as wires, in FPGA, also carry logic: a signal from one gate to another one has to run through the various *switches* which ensure the programmability of the routing network. This is one major difference when comparing FPGA and full-custom solutions.

The purpose of this paper is to study the long-term implications of this evolution. Section 2 will attempt to build a very rough but realistic model of the trend in FPGA technology demonstrated by Table 1. Section 3 then shows that this trend would lead to an increasing performance gap between FPGA and full-custom VLSI. The third section concludes by questioning the model, considering several possible evolutions of FPGA technology which would address the problems exposed here.

[1] The 4000 series contains 2 LUTs per configurable logic bloc (CLB). The Virtex series contains 4 LUTs per CLB. The LUT matrix is therefore given as (CLBs per line)×(CLBs per column)×(LUT per CLB).

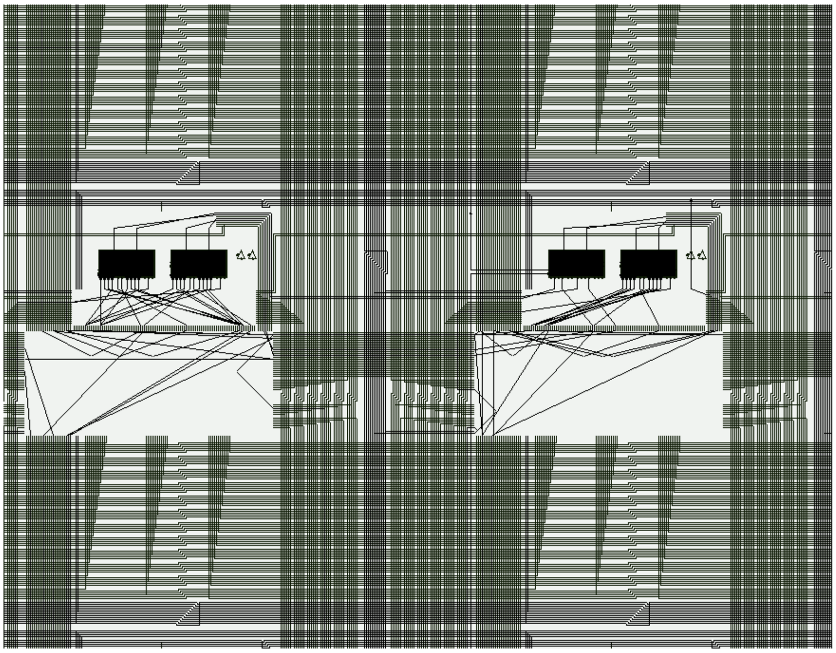

Figure 2: A close-up on Xilinx Virtex chip routing resources in the EPIC tool. The black boxes are computing units, the rest is wiring. Here again, routing switches are not shown, but the white boxes around the computing units are their input/output crossbars.

2 Definitions and hypotheses

2.1 Very large scale integration

The object of our study will be the evolution of FPGA computing power with respect to VLSI integration. To quantify this integration, we will use a universal measure, the *typical length* λ of a given VLSI process. Currently between $0.5\mu m$ and $0.1\mu m$, this length is related to the size of the smallest possible transistor. We shall express the various quantities we study as a function of λ.

As we study *integration*, we will consider circuits of fixed size, say square circuits of unit size[2].

As λ measures the size of a transistor, the *number of transistors* on this unit size chip grows as $O(\lambda^{-2})$. Besides, integration increases the *frequency* of the chip. As a very rough estimate we may assume that the switching time of a transistor is proportional to its width, thus to λ. Therefore the typical frequency

[2] It should be noted that the typical commercial size of a VLSI chip has been almost stable over the decade, at a few square centimeters. For example, the die size of the Intel Pentium II is roughly that of the 80286, even though intermediate processors of the same company have been bigger. In any case, the growth of die size doesn't compare to the growth of $1/\lambda$.

will grow as $O(\lambda^{-1})$, and the maximum raw power of the unit size chip grows as $O(\lambda^{-3})$.

This is only an upper bound on computing power: it assumes that all the silicon is used for transistors, whereas in any circuit, you have of course to dedicate some silicon to wires as well.

These hypotheses are linked to Noyce's thesis (as cited by Vuillemin [Vui94]) according to which λ is reduced by a constant factor $\alpha \approx 1.25$ every year. This integration leads to an increase of theoretical computing power of $\alpha^3 \approx 2$ every year. Vuillemin concludes from this that the peak FPGA performance also doubles every year, which we question in the following.

2.2 Field-programmable gate arrays

Let us now observe a family of theoretical FPGAs which closely follows VLSI integration: we will note FPGA(λ) the FPGA built in process λ. More precisely, FPGA(λ) has the following characteristics.

- We will assume that the architecture of the computing unit doesn't depend on integration. This is a very strong assumption, to be discussed later in section 4. We motivate it by the following observation: the base block of this computing unit, a 4-input look-up table, is the main common characteristic of the mainstream FPGA architectures from the two major FPGA vendors, Xilinx and Altera, and has been for almost a decade now.
- Let us assume this computing unit is a square of size $X_l\lambda$.
- Wires also have a width which, expressed in λ, is a constant, say $X_w\lambda$. In other terms you can have X_w/X_l wires in the width of a logic bloc, and this value doesn't depend on λ.

The last assumption may be discussed, considering that technology allows for more and more metal layers to put wires in (current FPGAs use 5 metal layers [Xil98]). The point is, however, that any FPGA wire must have some logic on it: in FPGAs, wires begin and end with *switches* which ensure the programmability of the routing, and these switches are built out of transistors, and thus consume transistor space.

We now need to quantify the number of wires we have in FPGA(λ).

2.3 A simple model of routing

It is impossible to give a model of all existing and possible routing architectures, therefore we will restrict the study to quantitative aspects.

Let $N(\lambda)$ be the number of LUTs per row or column of FPGA(λ). In [Vui94], Vuillemin neglects the space used by routing resources, and therefore assumes that $N(\lambda)$ grows as $O(\lambda^{-1})$. This assumption has been proven wrong (see table 1): the number of wires per LUT tends to increase with $N(\lambda)$. As a consequence, $N(\lambda)$ is no longer proportional to λ^{-1}.

Now we have to quantify this increase of routing. Table 1 shows that routing per channel grows almost linearly with N. We will therefore consider a number of wires per LUT (in other words a number of wires per routing channel) which is proportional to $N(\lambda)$ (let k be the proportionality factor). This corresponds to a Rent's exponent of 1. However the same results hold for values of the exponent between 0.5 and 1, as the interested reader may find in appendix. We choose here to keep the model simple.

3 The price of programmability

3.1 Spatial cost

Let us first estimate $N(\lambda)$. The width of an FPGA tile is now $(X_l+k.N(\lambda).X_w)\lambda$. As $N(\lambda)$ increases we soon have $X_l \ll k.N(\lambda).X_w$. Note that this is already the case in current FPGAs as shown by Fig. 2. In this figure, the two black boxes are two four-input lookup tables each, and the rest consists of routing and routing switches. Beware that this picture is a logical view from the Epic tool, and not an actual photograph of the layout: as such it might not perfectly reflect the scale of the various components, however it makes the point.

We now have an idea of the evolution of $N(\lambda)$ with respect to λ:

$$N(\lambda) \approx \frac{1}{k.N(\lambda).X_w.\lambda} \quad ,$$

hence

$$N(\lambda) \approx \sqrt{\frac{1}{k.X_w.\lambda}} \quad .$$

Lemma 1. *The number $N(\lambda)$ of LUTs in a row of FPGA grows as $O(\lambda^{-1/2})$.*

In other words, in order to double the number of LUTs on a row, we have to wait until λ has been divided by 4. Or: an FPGA will have twice the number of LUTs per surface unit only when the technology allows four times as many transistors.

3.2 Time cost

The previous analysis also has implications on the maximal operating frequency of FPGA(λ). Its inverse, the period, is the sum of two terms:

- The switching time for a logic bloc, which in our approximation is proportional to λ.
- The time to go through routing, which in turn is the sum of two terms:
  - A term which is at least proportional to the distance between the LUTs connected. This value is difficult to estimate, as it depends widely on the application. However this term is victim of the evolution of the distance between two LUTs, which is $1/N(\lambda)$, and as such evolves at best as $\lambda^{1/2}$;
  - A term which measures the time needed to go through routing switches. This term is the traversal time for one switch (proportional to λ, still in the same approximation), multiplied by the the number of switches on a wire. Note that these switches don't actually switch, except at configuration time, so their contribution is not dependent on the switching time of a transistor. However, transistors have a higher resistance than the wires between them, and we may consider that their traversal time evolves as

λ. The number of switches on a net, assuming a sensible routing algorithm, is at least $\log_2 N(\lambda)$, in which case the corresponding term in the period is $\lambda.\log_2 N(\lambda)$ and can be neglected, and at most $N(\lambda)$, in which case this term evolves as $\lambda.\lambda^{-1/2} = \lambda^{1/2}$ just like the previous one.

Finally, as λ decreases, the most significant term evolves in $\lambda^{1/2}$:

Lemma 2. *The frequency $F(\lambda)$ of $FPGA(\lambda)$ grows as $O(\lambda^{-1/2})$.*

3.3 Peak computing power

And finally the peak computing power grows as $N(\lambda)^2 \times F(\lambda)$, that is to say as $O(\lambda^{-3/2})$.

If for example λ is divided by 1.25 every year (following Noyce's thesis), the theoretical computing power of our family of FPGAs will double every other year, whereas that of pure VLSI will double every year. This means that the performance lag between FPGA and VLSI will increase (doubling every other year).

A sensible objection to our study is that we don't compare like for like, as we compare an FPGA with routing to VLSI without routing. This is certainly true: we are comparing peak, or theoretical, computing power. The point is that there exist very efficient architectures with limited or localized routing which, implemented in FPGA, will not benefit from this economy in routing when compared to a VLSI implementation.

Consider for example a microprocessor, and suppose that the critical component of its arithmetic unit is, say, a multiplier. This component will therefore be heavily optimized for performance. The designers of this multiplier won't for example allow any external routing channel through it: external routing will flow around the multiplier (or above if there are metal layers available). Now imagine the same processor, with the same multiplier, implemented in FPGA(λ): these optimizations are no longer possible, because routing channels, most of which are used by external routing, cut through the multiplier.

This is, once again, probably the main difference between FPGA and full-custom VLSI: in full-custom one pays the performance cost of the routing one gets, whereas in FPGA one pays the cost of all the potential routing even if it isn't being used.

4 Discussion

This study is not meant to be pessimistic: we trust FPGA vendors for contradicting this dooming view in the forthcoming years. To do so, they just have to contradict one of our hypotheses. Note that, as shown in appendix, taking into consideration a Rent's exponent smaller than 1 doesn't change our conclusion (as long as $r > 0.5$, which is a prerequisite for an FPGA).

4.1 Constant granularity

We have supposed that the size of the basic building block will remain constant, while their number grows. However it is to be expected that the size of the computing unit will grow as well, in order to keep a balance in the routing resources. Note that it is the case in the evolution form the 4000 series to the Virtex series, as the size of the configurable logic bloc has increased from 2 to 4 LUTs. Another example of this trend is Chess [MSK+99], a reconfigurable array aimed at multimedia applications, where the unit of both computing and routing resources is 4 bits. This architecture is closer in many respects to the MasPar massively parallel computers [HPK95] than to mainstream FPGAs.

It is difficult here to resist comparison with the history of parallel computing: the number of processors in parallel computers doesn't grow anymore. Today's biggest parallel computer have no more than a few hundreds of processors, much less than their predecessors of the previous decade (MasPar and TMC computers had tens of thousands). The growth is now in the performance of each processor. We should expect the same kind of evolution in the FPGA world.

Exploring this parallel further, let us notice that increasing emphasis, in the parallel computing community, is set on architectural and system techniques designed to hide both the complexity and inefficiency of the communications (e.g. distributed shared memory and caches). Could this trend not filter down to the FPGA world? This is obviously not only a question of architecture.

4.2 Regular architecture

We also expect more and more hierarchy in the FPGA structure [RH97]: near-linear routing is probably not necessary at any scale.

Attempts to hierarchical architectures have already been made, for example in the Teramac system [SKC+95], or in the Xilinx Series 6000 which had a logarithmic routing architecture. The newest Actel FPGAs [KBK+99] claim to be "semi-hierarchical", with three levels of hierarchy, each level being the usual mesh.

However the Xilinx 6000 series was a commercial failure. To say that it was due to its hierarchical routing would be exaggerating: its main weakness, in our sense, was in its vendor place-and-route tools. Altough they were very good at allowing the designer to organize a hierarchy of subcircuits, they were very bad at doing anything automatically.

What this example makes clear is that hierarchical routing architectures need sounder bases on the software side to really succeed. This is only one aspect of the evolutions in FPGA support software needed to address the problems shown in this paper.

4.3 FPGA synthesis tools

Current FPGA synthesis tools discard the logical hierarchy of a circuit to perform a global optimization of the place and route problem. In the VLSI world, this approach has long been replaced with a more hierarchical one, where a big circuit is decomposed in smaller functional blocks which are then optimized locally (or found in libraries).

This doesn't only ease the task of the designer, but also leads to efficient architectures by breaking up global optimization problems into tractable, local ones.

This point is obviously linked to the previous one: for the hierarchical approach to have the same performance benefits in FPGAs, there must be some form of hardware hierarchy. This was the case in the Xilinx 6000 series.

4.4 General purpose FPGAs, or not

Some classes of applications don't need general purpose routing, and would be contented with more scalable architectures. An example is that of datapath-oriented FPGAs, as studied in the Garp project [HW97] among other. Chess [MSK+99] is dedicated to multimedia processing, and this loss of generality allows for a more efficient architecture in its application field.

An even more radical example is the class of systolic arrays, as obtained using automatic parallelization techniques [Qui84, dD97]: these techniques yield designs with only clock and local routing, which scale well. Could a FPGA family with constant, local routing dedicated to systolic applications be designed ? This class of applications, unfortunately, is probably too restricted to make a such a systolic FPGA commercially viable.

5 Conclusion

The increase of road traffic in expanding cities is a fatality for complexity reasons very similar to those presented here. Most big cities have tried to address this traffic increase by widening the roads and increasing the number of lanes. Since the eighties, however, this approach to city traffic is generally considered a failure. One reason is specific to the city metaphor: city planners, unlike FPGA designers, can't move blocks further apart when they want more lanes between them. But even if they could, they would face the citizens' critics: the useful part of the city is the blocks, that is where people live and interact. City-residents are increasingly reluctant to sacrifice this living space to traffic lanes, all the more as they are conscious of the short lifespan of any traffic improvement.

Similarly, this study showed that the current evolution of FPGAs gives an increasing proportion of the silicon resources to routing, which is computationnaly useless. It establishes that this trend, if pursued, would lead to an increase of the performance gap between FPGA and full-custom VLSI. The hope is to push long-term researchers to explore alternative possibilities.

For example, some cities manage their traffic increase quite well by promoting public transportation (the metaphor of a bigger granularity), local shopping and working (the metaphor for architectural solutions placing the emphasis on local routing), and cycling.

Alas, as far as cycling is concerned, we couldn't find an FPGA metaphor.

Acknowledgements

Special thanks should go to Wayne Luk, Nabeel Shirazi and the ALA team at Imperial College, London, and to Dominique Lavenier of IRISA, Rennes, for many interesting discussions and comments. This work was partly supported by an INRIA post-doctoral fellowship at Imperial College, London, UK.

References

[Bre98] Gordon Brebner. Field-programmable logic: Catalyst for new computing paradigms. In *International Workshop on Field Programmable Logic and Applications*, Tallin, Estonia, September 1998.

[dD97] Florent de Dinechin. Libraries of schedule-free operators in Alpha. In *Application Specific Array Processors*. IEEE Computer Society Press, July 1997.

[DeH94] Andre DeHon. DPGA-coupled microprocessors: Commodity ICs for the early 21st century. In *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994.

[DeH96] Andre DeHon. *Reconfigurable Architectures for General-Purpose Computing*. PhD thesis, MIT, August 1996.

[HPK95] M. Hamdi, Y. Pan, and W. T. Kwong. Efficient image processing applications on the maspar massively parallel computers. *International Journal of High Speed Computing*, 7(4):489–514, 1995.

[HW97] John R. Hauser and John Wawrzynek. Garp: a MIPS processor with a reconfigurable coprocessor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 12–21, Napa Valley, CA, April 1997.

[KBK+99] S. Kaptanoglu, G. Bakker, A. Kundu, I. Corneillet, and B. Ting. A new high density and very low cost reprogrammable FPGA architecture. In *FPGA'99, ACM/SIGDA International Symposium on FPGAs*, pages 3–12, Monterey, CA, February 1999.

[MSK+99] A. Marshall, T. Stansfield, I. Kostarnov, J. Vuillemin, and B. Hutchings. A reconfigurable arithmetic array for multimedia applications. In *FPGA'99, ACM/SIGDA International Symposium on FPGAs*, pages 135–143, Monterey, CA, February 1999.

[Qui84] P. Quinton. Automatic synthesis of systolic arrays from recurrent uniform equations. In *11th Annual Int. Symp. Computer Arch., Ann Arbor*, pages 208–214, June 1984.

[RH97] J. Rose and D. Hill. Architectural and physical design challenges for one-million gate FPGAs and beyond. In *FPGA'97, ACM International Symposium on FPGAs*, pages 129–132, Monterey, CA, February 1997.

[RM98] B. Radunovic and V. Milutinovic. A survey of reconfigurable computing architectures. In *International Workshop on Field Programmable Logic and Applications*, Tallin, Estonia, September 1998.

[SKC+95] G. Snider, P. Kuekes, W. B. Culbertson, R. J. Carter, A. S. Berger, and R. Amerson. The Teramac configurable computer engine. In *Field Programmable Logic and Applications*, pages 44–53. LNCS 975, September 1995.

[TMM+98] A. Takahara, T. Miyazaki, T. Murooka, M. Katayama, K. Hayashi, A. Tsutsui, T. Ichimori, and K. Fukami. More wires and fewer LUTs: a design methodology for FPGAs. In *FPGA'98, ACM/SIGDA International Symposium on FPGAs*, pages 12–19, Monterey, CA, February 1998.

[Ull84] Jeffrey D. Ullman. *Computational Aspects of VLSI*. Principles of Computer Science. Computer Science Press, 1984.

[Vui94] Jean Vuillemin. On computing power. In *Programming Languages and System Architectures, LNCS 782*, pages 69–86, Zürich, Switzerland, June 1994.

[Xil97] Xilinx Corporation. *XC4000E and XC4000X Series Field Programmable Gate Arrays*, November 1997.

[Xil98] Xilinx Corporation. *Virtex 2.5V Field Programmable Gate Arrays*, October 1998.

A When Rent's exponent is smaller than 1

This section follows the computation of section 3 in the case of a Rent's exponent between 0.5 (constant width channels) and 1.

A.1 Spatial cost

As exposed in the introduction, a Rent's exponent r corresponds to a channel width of $2r-1$. The width of an FPGA tile is thus $X_l + k.N(\lambda)^{2r-1}.X_w$, and we still have $X_l \ll k.N(\lambda).X_w$, leading to

$$N(\lambda) \approx \frac{1}{k.N(\lambda)^{2r-1}.X_w.\lambda} \quad ,$$

hence

$$N(\lambda) \approx \sqrt[2r]{\frac{1}{k.X_w.\lambda}} \quad .$$

Lemma 3. *The number $N(\lambda)$ of LUTs in a row of FPGA grows as $O(\lambda^{-\frac{1}{2r}})$.*

A.2 Time cost

The period is the sum of:

- The switching time for a logic bloc, proportional to λ.
- The time to go through routing, which in turn is the sum of two terms:
  - A term expressing the time to go through the wires themselves (i.e. to load the corresponding capacitor), at least proportional to the distance between the connected LUTs; two LUTs are at least separated by one routing channel, thus the minimal distance between two LUTs is $1/N(\lambda)$. This term evolves therefore at best as $\lambda^{\frac{1}{2r}}$;
  - A term which measures the time needed to go through routing switches. This term is the traversal time for one switch (proportional to λ), multiplied by the the number of switches on a wire. The number of switches on a net, assuming a sensible routing algorithm, is at least $\log_2 N(\lambda)$, in which case the corresponding term in the period is $\lambda.\log_2 N(\lambda)$ and can be neglected, and at most $N(\lambda)$, in which case this term evolves as $\lambda.\lambda^{-\frac{1}{2r}} = \lambda^{1-\frac{1}{2r}}$. As $1/2 < r < 1$, we have $0 < 1 - \frac{1}{2r} < \frac{1}{2r}$. As λ decreases, the most significant term in the period is $\lambda^{1-\frac{1}{2r}}$ in this case. However this term also assumes a very bad routing architecture, in which the number of switches on a typical net is proportional to the size of the chip. This is what you would get on an FPGA with only local routing, for instance.

Let us be optimistic and assume that the routing is good enough for the period to be limited by the second term in $\lambda^{\frac{1}{2r}}$. We get the following optimistic asymptotic frequency:

Lemma 4. *The frequency $F(\lambda)$ grows as $O(\lambda^{-\frac{1}{2r}})$.*

A.3 Peak computing power

And finally the peak computing power grows as $N(\lambda)^2 \times F(\lambda)$, that is to say as $O(\lambda^{-\frac{3}{2r}})$.

This result yields the same conclusions with respect to an increase of the time lag between full-custom VLSI and FPGA peak computing power.