Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Dynamic reconfiguration architectures for multi-context FPGAs

Yitzhak Birk, Evgeny Fiksman *

Electrical Engineering Department, Technion–Israel Institute of Technology, Haifa 32000, Israel

ARTICLE INFO

Article history:
Available online 31 January 2009

Keywords:
FPGA
Reconfigurable computing
Multi-context architectures

ABSTRACT

Field-programmable gate arrays (FPGAs) are being integrated with processors on the same motherboard or even chip in order to achieve flexible high-performance computing, and this may become main stream in chip multi-core architectures. However, the expensive FPGA area is often used inefficiently, with much of the logic idle at any given time. This work, motivated by the Dynamic-Link Library (DLL) concept in software, explores the possibility of "hardware DLLs" by finding ways for fast dynamic incremental reconfiguration of FPGAs. So doing would, among other things, enable same-function replication at any given time, with functions changing quickly over time, thereby enabling efficient exploitation of data parallelism at no additional hardware cost.

We present two new multi-context FPGA architectures based on two different configuration storage architectures: local and centralized. Problems such as configuration storage and reconfiguration (time, power and space) overhead are considered. Well known area and power models are used in evaluating various approaches and in order to provide guidelines for matching architectures to target applications. Lastly, we provide insights into resulting scheduling issues. Our findings provide the foundation and "rules of the game" for subsequent development of reconfiguration schedulers and execution environments.

1. Introduction

Field-programmable gate arrays (FPGAs) are useful for flexible high-performance computing. As part of the general "System on-Chip" trend, there is a clear trend to combine the implementation efficiency of general purpose CPUs and the ultimate power-performance of ASICs for dedicated tasks at the extremes, with FPGAs covering the middle ground, namely specific tasks that are not known in advance. One example is the Xilinx® Virtex®IV devices that integrate custom logic, fixed DSP processors, network cores and even general purpose CPUs. Others are the AMD® Opteron™ and Intel® Xeon™ dual-processor motherboards, in which one of the two processor slots can be populated with an FPGA [24].

These "mixed" architectures feature great flexibility, but many applications do not use the configurable "FPGA" part efficiently because different functions are required at different times. This is the case with applications that require many different functions yet, due to their own precedence constraints, can only use some subset of these functions at any given time. The result is a need for a large, expensive FPGA, and at times even impossibility. Moreover, with leakage power becoming a major contributor to total power consumption, there is also a major (and growing) power penalty. In a study of area efficiency of common, single context FPGA [7], the authors show that for more than twenty circuits from the MCNC benchmarks, average area efficiency is merely 19%. With their multi-context circuits, in contrast, the average area efficiency is above 70%.

* Corresponding author. Tel./fax: +972 54 5415017.
E-mail addresses: birk@ee.technion.ac.il (Y. Birk), evgeny.fiksman@intel.com (E. Fiksman).

0045-7906/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2008.11.024

Nomenclature

FPGA	Field Programmable Gate Array
MC-FPGA	Multi-Context FPGA
DLL	Dynamically Loaded Library
NMOS	n-type metal-oxide-semiconductor field effect transistors
PMOS	p-type metal-oxide-semiconductor field effect transistors
CMOS	Complementary Metal Oxide Semiconductor
ASIC	Application Specific Integrated Circuit
SRAM	Static Random Access Memory
NROM	A unique localized charge-trapping-based non-volatile memory technology that employs two separate physical charge packets, realizing two and four bits per cell
ITRS	International Technology Roadmap for Semiconductors, an international body for guiding the semiconductor industry
DIBL	Drain-Induced Barrier Lowering
PSRAM	Pseudo-Static Random Access Memory
DRAM	Dynamic Random Access Memory
CLB	Configurable Logic Block
CLC	Configurable Logic Cluster
GPP	General Purpose Processor
MUL	Multiplier
LUT	Lookup Table
D-FF	Data Flip Flop
SOPC	System On programmable Chip
JTAG	Joint Test Action Group, also known as Boundary-Scan
T_{OX}	Oxide thickness
V_{th}	Threshold voltage [*Volt*]
V_{DD}	Supply voltage[*Volt*]
A_x	Estimated area [NMOS transistors]
$\hat{K}$	Estimated power coefficient [natural]
[ ]	Rounding operation
$\lceil\ \rceil$	Ceil operation
e	Exponent exp(1)
λ	Feature size
W	Transistors' width
L	Transistors' length
C_n^k	Is a "choose function" /binomial coefficient $C_n^k = \binom{n}{k} \frac{n!}{k!(n-k)!}$

To illustrate the point, consider the message-digest MD5 algorithm [RFC1321]. It comprises four basic functions (F, G, H and I) that are executed serially (Fig. 1). The algorithm operates on 512-byte data chunks, and requires 64 loop iterations for the hash calculation. Also, the input to function 'F' depends on function I's output from the previous iteration. Therefore, only one function can be active at any given time and the algorithm cannot be pipelined. Consequently, only 25% of a device that holds all functions are active at any given time, resulting in area inefficiency and high static power consumption. The need to implement all types of logic blocks at all times moreover limits the exploitation of available data parallelism, as the number of same-function blocks is severely limited.

The above deficiency can be addressed by the hardware equivalent of dynamically linked libraries (DLL) used in software. With this, one would be able to dynamically reconfigure parts of the FPGA while others are operating, such that at any given time, some of the functions are active while others are inactive and are merely stored. We refer to this as a *Multi-Context FPGA* (MC-FPGA). Our goal in this paper is to identify and address some important elements of such an approach. One potential benefit, particularly for high-performance computing, is that at any given time one can implement more instances of a given function, thereby permitting much greater exploitation of available data parallelism.

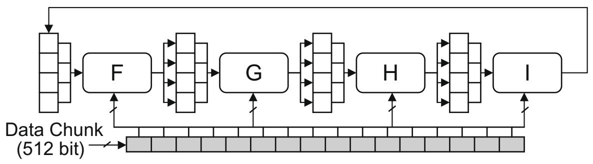


Fig. 1. MD5 algorithm flow.

An MC-FPGA has active contexts and a number of standby configurations. To be truly useful, one must furthermore partition the FPGA into configuration regions, such that some can be reconfigured while others are working. In the MD5 example, area and power models presented in this paper show that approximately 70% of the area is saved by using a single active configuration and three standby ones, and static power is reduced by some 60%.

An interesting problem related to MC-FPGA is the time/space scheduling of configurations and their assignment to regions. However, given the simple nature of configuration blocks, one should first address the question of whether the area and power required for supporting standby configurations (memory, multiplexers, routing paths, etc. and the power consumed by reconfiguration) are smaller than those required for permitting all contexts to be active. Therefore, the focus of this work is on critical circuit and architecture issues that must be addressed in order to make the MC-FPGA approach viable. These include the placement of stored standby configurations (centralized or distributed), the type of memory cells (e.g., flash or volatile), and the required control circuitry. Area, power and speed implications as well as some scheduling implications are considered.

Evaluation of single context FPGAs revealed that configuration-related items occupy about 10% of total area and consume about 3% of total leakage power. These numbers suggest that, if done properly, reconfiguration overhead may be reasonable, giving us hope.

The contributions of our work are mostly at the circuit/logic design and architecture levels. We propose an efficient circuit for access to small NROM [3] based memory units that normally require complex access circuitry. Also, we propose two viable and efficient architectures for multi-context FPGAs. Finally, we present the opportunity for inter-region configuration swaps and provide insights pertaining to its usefulness.

The remainder of the paper is organized as follows. Section 2 surveys related work. Section 3 provides a brief overview of the area and power models used in this work. (Appendix A provides more detail.) Section 4 presents our local storage approach, wherein the inactive configuration data is stored beside each active cell. (This general approach is not novel, and several works discuss the use of CMOS [4] and DRAM [5] cells for this purpose, but we present a different technique based on dual bit NROM cells. The use of such cells reduces both the area and the static power required for inactive configuration storage, but requires different control.) In Section 5, we continue with our novel hybrid storage architecture, wherein only a single "standby" configuration is stored alongside each active logic block, while the rest of the configuration data is stored centrally outside the active area. In Section 6 we compare these architectures, and provide guidelines for their suitability to different applications. In Section 7 we discuss some scheduling issues, and Section 8 offers concluding remarks along with directions for further research.

2. Related work

First steps toward commercial dynamically reconfigurable FPGA were taken in the Xilinx® VirtexII™ [6] family, wherein part of the device could be reconfigured while another part is running. Such devices are called partially reconfigurable. However, new configurations must be loaded into the chip, resulting in very high reconfiguration latency. This process is moreover power inefficient due to configuration protocol complexity.

A number of multi-context architectures were proposed during the last decade. In [5], the authors present a DRAM based multi-context architecture wherein additional configurations are stored locally. Benchmarking results used in order to show the impact of multi-context technology show 3–4*x* improvements in resource utilization [7]. However, the implications on static power consumption are ignored. From an architecture perspective [5], the use of DRAM memory for the local configuration storage requires complex, power-hungry and area inefficient circuitry.

In work by Xilinx [4], the standard XC4000E FPGA device was extended to a multi-context one. This device used CMOS cells, and permitted eight contexts. Entire chip reconfiguration was supported. Unfortunately, the area and power issues were not addressed.

Another work [8] deals with a centralized configuration storage architecture, and demonstrates fast context switching. However, the proposed context switch circuitry is very complex. There too the authors did not address the area and power issues.

Table 1 summarizes multi-context methods and their implication on reconfiguration time, on-chip area overhead and consumed power. Both previously proposed approaches and our proposals (the bottom two) are included.

Table 1
Multi-context FPGA methods (*our proposals).

Design	Multi-context method	Reconfiguration time	Area overhead (on-chip)	Additional static power (on-chip)
VirtexII™ [6]	Off-chip context switching	Very long	No	No
DPGA[5]	On-chip, distributed memory	Short	High	High
Time-multiplexed FPGA[4]	On-chip, distributed memory	Short	Very high	Very high
Sanders FPGA[8]	On-chip, centralized memory	Medium	High	High
MC-FPGA* (local)–NROM	On-chip, distributed memory	Short	Medium	Low
MC-FPGA* (global)–LPDRAM	On-chip, centralized memory	Medium (with prefetch)	Low	Low

Additional works explored elements of the multi-context approach. For example, [9] studies a configuration memory fragmentation problem and proposes a de-fragmentation algorithm for incrementally reconfigurable multi-context architectures.

The multi-context approach is not restricted to the FPGA domain. In [10], the authors present arithmetic multi-context reconfigurable arrays based on local configuration storage, using a programmable FIR with dynamically varying coefficients as an example.

3. Power and area evaluation models

This section describes the power and area models used in our comparative evaluation of our proposed architectures and a baseline architecture. The derivation of power and area estimates for an entire architecture using these models is illustrated in Appendix A.

3.1. Power model

There are two common forms of power consumption in digital circuitry: dynamic and static power. Dynamic power consumption occurs during state transition, caused by charging of sequential transistor nodes ("switching power") and by short-circuit current during gate transition ("short-circuit power"). Unlike dynamic power, static power occurs in transistor idle state and is caused by leakage currents within the transistor.

The dynamic power consumption was dominant for many years, and much effort was spent in order to reduce it. However, as technology advanced to feature sizes of 90 nm and below, static power has become equal to dynamic and even exceeds it. Fig. 2 depicts the total chip dynamic and static power dissipation trends based on the International Technology Roadmap for Semiconductors [11]. The ITRS projects a decrease in dynamic power per device over time. However, if we assume a doubling of on-chip devices every two years, total dynamic power will increase on a per-chip basis. Projections of the static power are different. While the sub-threshold leakage power continues to grow, the gate-oxide leakage becomes less significant. This projection is supported by the latest information released by the Intel® Corporation [1]. The new Hafnium-based 45 nm process ("High-K" transistors) provides a greater than 10-fold reduction in gate-oxide leakage compared to the previous 65 nm process.

The leakage power is the product of the supply voltage and leakage current:

$$P_{static} = V_{dd} \cdot (I_{sub} + I_{ox}) \tag{1}$$

where V_{dd} is the supply voltage, and I_{sub} and I_{ox} are the sub-threshold and gate-oxide leakage currents, respectively.

In most situations, the total number of logic-level transitions required for reconfiguration is much smaller than the number of such transitions between reconfiguration events. The average dynamic power required by the reconfiguration process itself can thus be neglected, so our focus is on the leakage power. Assuming the use of high-K transistors [1], whose gate leakage current is negligible [2], we moreover only consider sub-threshold leakage ($I_{ox} = 0$).

We are only interested in a high level evaluation, so we use the high level sub-threshold power model of [13], which derived from spice MOS transistor model [14], namely

$$P_{static} = V_{dd} \cdot N \cdot k_{design} \cdot \hat{I}_{leak}. \tag{2}$$

The description of model parameters is given in Table 2.

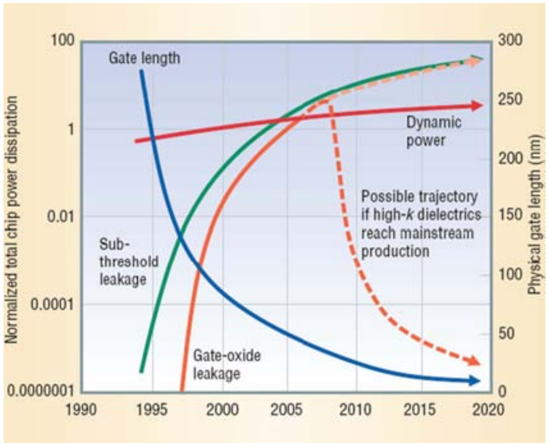


Fig. 2. Power-consumption trends [12].

Table 2
Static power model parameters.

Parameter	Description
V_{dd}	Power supply voltage
N	The number of transistors in the design
k_{design}	Process independent; represents the characteristics of an average device. Encompasses the distribution of device types (N- and P-type), geometries (W/L), states (on/off), and transistor's stacking factors. Obtained by manual analysis or empirically from circuit simulation
$\hat{I}_{leak}$	Technology parameter describing the device sub-threshold leakage, expressed in Eqs. (3) and (4). Architecture agnostic!

Table 3
k_{design} values.

Circuit	N	k_{design}	Notes
D flip–flop	22/bit	1.4	Edge triggered FF
D latch	10/bit	2.0	Transparent latch
2-input mux	2/bit/input	1.9	+1.2/input over 2
6T RAM cell	6/bit	1.2	1 RW port
Static logic	2/gate input	11	Depends on speed, load (±3)

This model hides the technology process parameters in $\hat{I}_{leak}$, which is expressed in Eq. (3). The value of $\hat{I}_{leak}$ is architecture agnostic and will thus not affect our comparisons, so we ignore it.

$$\hat{I}_{leak} = I'_{s0} \cdot \left(1 - e^{\frac{-V_{ds}}{v_t}}\right) \cdot e^{\frac{V_{gs}-V_T-V_{off}}{n \cdot v_t}}, \tag{3}$$

where I'_{s0} and V_{off} are empirically determined model parameters, v_t is a physical parameter proportional to the temperature, n is derived from a host of other model and device parameters, V_T is the transistor threshold voltage, and V_{gs} and V_{ds} are gate-source and gate-drain voltages.

Hand calculated k_{design} parameters for the common designs are given in [13] and presented in Table 3. The table also lists the number of transistors (N) used in reference circuits for calculating the k_{design} values along with notes about adjustments required in k_{design} when applied to specific circuits.

These values will be used for the static power evaluation in our work. We will assume that the same technology parameters are used in various implementations, so $\hat{I}_{leak}$ will remain as a constant value. We will derive a power coefficient $\hat{K}_{arch}$ parameter that will be used for comparative evaluation of different implementations. The $\hat{K}_{arch}$ parameter combines k_{desing}, N and W/L aspect ratio, and will be calculated separately for each one of the architectures. Eq. (4) expresses the relation between the above parameters and $\hat{K}_{arch}$.

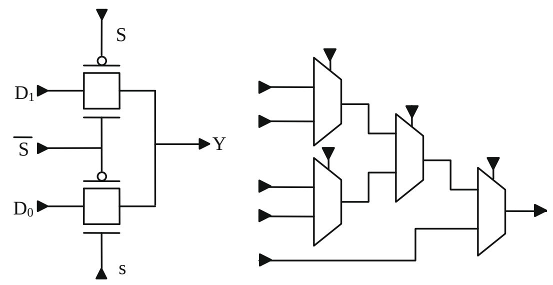
$$\hat{K}_{arch} \propto f(W/L, N, k_{design}) \tag{4}$$

The updated static model is expressed:

$$P_{static} = V_{dd} \cdot \hat{K}_{arch} \cdot \hat{I}_{leak} \tag{5}$$

3.2. Area model

The die size of a typical FPGA is dominated by transistor area [15]. We base our area evaluation on the area estimation model initially proposed in [15] and lately used in [16,17]. It estimates the total area required to implement a desired architecture, expressed in units of the minimum-width NMOS transistor ($W = L = \lambda$) area. We assume that the area of a PMOS transistor, which satisfies symmetric rise and fall times of standard NOT gate, is 3.5 times that of an NMOS transistor [18].



(a) 2→ 1 Selector layout (b) 5→ 1 Selector schematics

Fig. 3. Selector (MUX) layout.

3.3. Example – selector (multiplexer) area and power models

A selector chooses one of its input pins based on the values of its indexing pins. Usually, $N_{Inputs} = 2^{Nindex}$. The basic building block is a $2 \rightarrow 1$ selector (Fig. 3a), comprising two NMOS and two PMOS transistors. Its total equivalent area is $A_{2 \rightarrow 1} = 2 \cdot 1 + 2 \cdot 3.5 = 9$ minimal transistors.

In order to build larger selectors, a "tree" of $N_{2 \rightarrow 1}$ $2 \rightarrow 1$ blocks is used, where

$$N_{2\to1} = \left[\sum_{i=1}^{\lceil \log_2(N_{input}) \rceil} \left\{\frac{N_{input}}{2^i}\right\}\right] = \left[N_{input} \times \left(1 - \left(\frac{1}{2}\right)^{\lceil \log_2(N_{input}) \rceil}\right)\right], \tag{6}$$

$$A_{MUX}(N) = A_{2\to1} \times \left[N \times \left(1 - \left(\frac{1}{2}\right)^{\lceil \log_2(N_{input}) \rceil}\right)\right] = 9 \times \left[N \times \left(1 - \left(\frac{1}{2}\right)^{\lceil \log_2(N_{input}) \rceil}\right)\right]. \tag{7}$$

As an example, the layout of a $5 \rightarrow 1$ selector is depicted in Fig. 3b. The power coefficient of the selector is based on the power coefficient model presented in Table 2 and is expressed

$$\hat{K}_{MUX}(N) = 2 \times N \times (1.9 + 1.2 \times (N - 2)). \tag{8}$$

In the following sections, we use the above models along with the logic of the various architectures to obtain estimates of their relative merits in terms of static power consumption and required area. In those evaluations, we focus on the actual memory cells and the configuration-change logic. Data paths are ignored, as they are largely unchanged relative to single context FPGAs (which also need to be configured). The logic required for execution of the reconfiguration scheduler (as opposed to the actual reconfigurations) is ignored, as it is likely to be negligible in view of the moderate number of modules and the likelihood that quite a few operations will take place between configuration changes.

4. Single context FPGA

This section describes the high level architecture of common (single context) FPGAs, and presents area and power evaluation results that constitute a baseline for our study. This evaluation serves two purposes: (1) training the proposed area and power models – get initial feelings and skills, and (2) validating our models versus previous knowledge.

4.1. FPGA architecture overview

A modern FPGA structure may be divided into three components: configuration memory, active logic and routing. Configuration memory and active logic are usually combined into small structures called a Tile or a Configurable Logic Block –

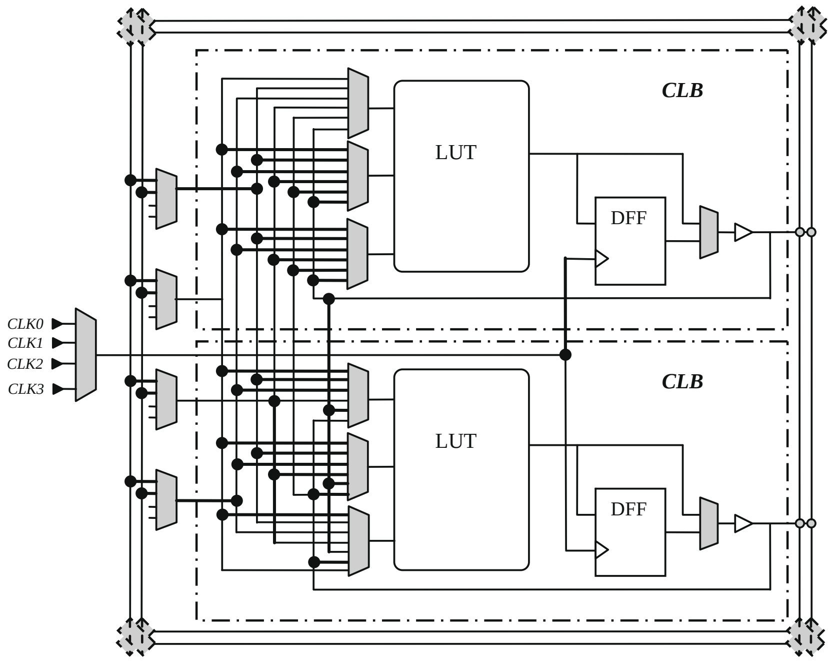


Fig. 4. Configurable cluster architecture.

CLB. Each CLB implements a user defined logic function, and is split into two parts: one comprises fixed logic units like FF, latch, MUX and additional combinational units; the other is reconfigurable, comprising memory components like CMOS cells. The reconfigurable memory components enable implementation of a general N-input logic function. The CLBs are arranged in groups, called Configurable Logic Cluster – *CLC*. Fig. 4 depicts a CLC comprising a pair of CLBs.

The results of the logic cluster calculations are distributed across the device by Signal Routers. Routers are generally based on bypass transistors that are active whenever two specific wires should be connected, and on tri-state buffers that distribute signals over long distances within a device. The actual routing configuration is stored in local memory cells.

The described style of FPGAs is known as an island style [15]. I/O units are located on the perimeter of an FPGA. Each CLB is connected to a number of surrounding wires. Each connection is flexible and depends on the specific configuration. Connection wires are divided into local and global wires. The local wires are gateways from the logic blocks and connect nearest routers. In contrast, the global wires connect distant routers and may pass over the chip. In large FPGAs, the local wires pass over a number of routers and may directly connect CLBs. The number of wires in an FPGA heavily affects chip performance. The island FPGA approach places some restrictions on the architectures that can be implemented; however, each tile can be replaced with a different logic block like a multiplier or even a compact General Purpose Processor (GPP). Fig. 5 depicts an island style FPGA with I/O Ports, configurable logic blocks, general purpose processors and multipliers.

4.2. Area and power evaluation results

The detailed model derivation is presented in [23] (and Appendix A.1). The major factor influencing FPGA performance and area is the *Cluster Size* (N_{CLB}), defined as a number of CLBs located inside a single CLC. This value is chosen through a tradeoff between internal (selectors) and intra-chip (cross chip wires) routing. The cross-wires are more expensive (area

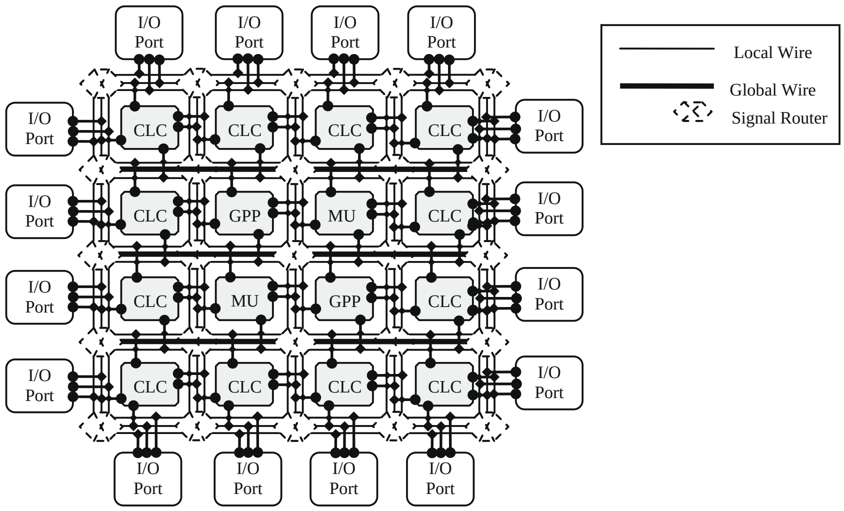


Fig. 5. Island style FPGA architecture.

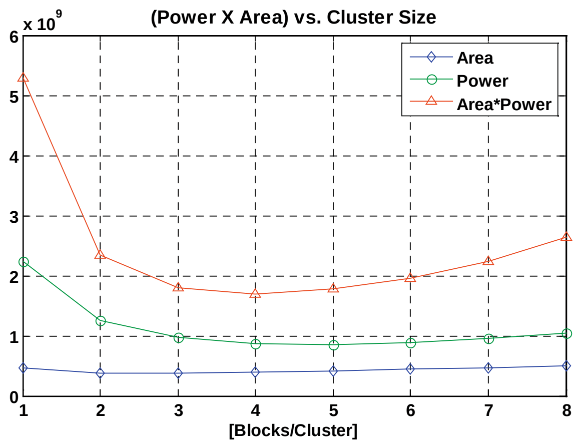


Fig. 6. Single context FPGA evaluation.

and power) than small selectors. The goal of the evaluation process is to estimate the optimal cluster size (N_{CLB}). In order to state the estimated FPGA performance by a single value, the product $A_{FPGA} \times K_{FPGA}$ is used (small is good). Fig. 6 depicts the overall device evaluation versus cluster size (N_{CLB}). For small N_{CLB}, the majority of the area and power are spent on the cross-chip wires. For $N_{CLB} > 4$, the internal selectors become dominant, and the number of intra-chip wires drops.

The combined measure suggests an optimal cluster size of 4 blocks. At this working point, 10% of the total area is occupied by the configuration bits, and 3% of total static power is consumed by the active configurations.

These results were compared with [15,16,19] that used a similar area model. For example, [16] presents an FPGA device wherein the optimal lookup table (LUT) size is 4 or 5, and the cluster size (N_{CLB}) varies between 4 and 9. The commercial VirtexII® devices [6] that are based on a similar architecture use $N_{CLB} = 4$. These facts suggest that our model is reasonable.

As our baseline configuration, we use an active matrix size of 192×116 blocks, $N_{CLB} = 4$, 4-input lookup tables (LUT) and an average wire length $s = 4$. (LUT size and wire length were adopted from previous studies [16,17].)

5. Local storage multi-context FPGAs

This section presents an approach for multi-context FPGA, which is based on local (''distributed'') storage as described in [4,5]. Here, the space for configuration bits is replicated as the number of desired configurations. These bits are distributed among the various reconfigurable units like CLC and Signal Router. In order to support a multi-context execution environment, the intermediate memory (Flip Flops) and internal routing selectors must also be replicated, as depicted in Fig. 7 for an extended multi-context CLC. Configuration changes are local, i.e., one of the locally stored standby configurations may become active. We consider two implementations of the local (distributed) storage: one based on CMOS cells, and the other based on area- and power-efficient NROM cells [3]. Note that the common wisdom of large memory chips is not relevant to storing additional configurations: the control read/write logic, which is normally used to serve numerous data cells, is used here for a very small number of cells and therefore is not negligible. One contribution of this work is a novel efficient NROM access circuit, described later.

Our goal is to investigate the impact of additional standby configurations on the total area and on static power. In the evaluation, the number of active cells remains fixed, and we vary N_c, the number of standby contexts per cell. The baseline configuration has the same parameters as described in Section 4.2. In order to evaluate the area and the power coefficient of the multi-context devices, the previously described single context FPGA model is extended. The highlights of the detailed model derivation are presented in Appendix A.2, and full details are provided in [23].

5.1. CMOS based local storage

The study shows that as the number of contexts (N_c) increases, storage of the additional configurations becomes inefficient. For example, when $N_c = 10$ the area occupied by additional configurations is 46% of the total area (Fig. 8a), and the additional consumed static power is about 10% (Fig. 8b).

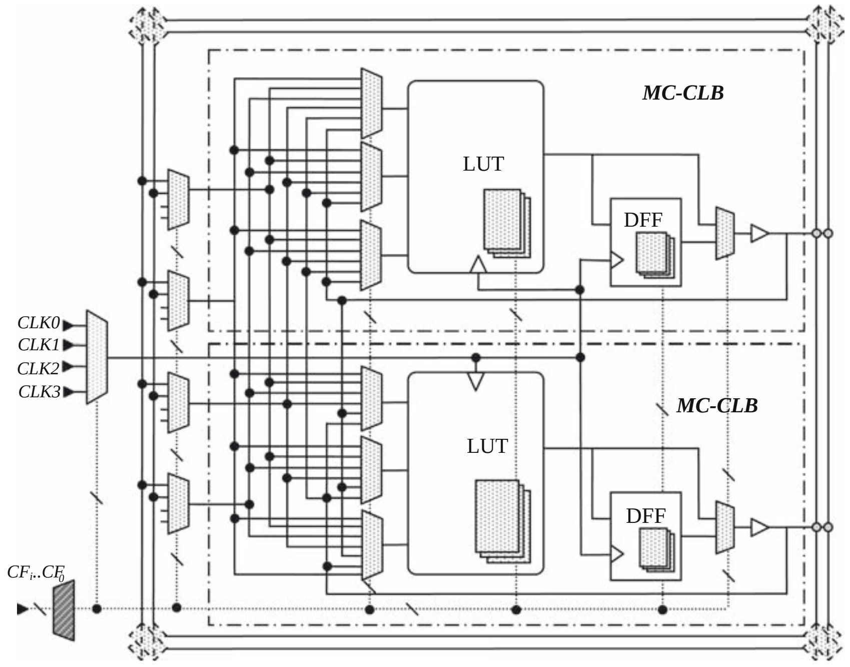


Fig. 7. MC-cluster architecture.

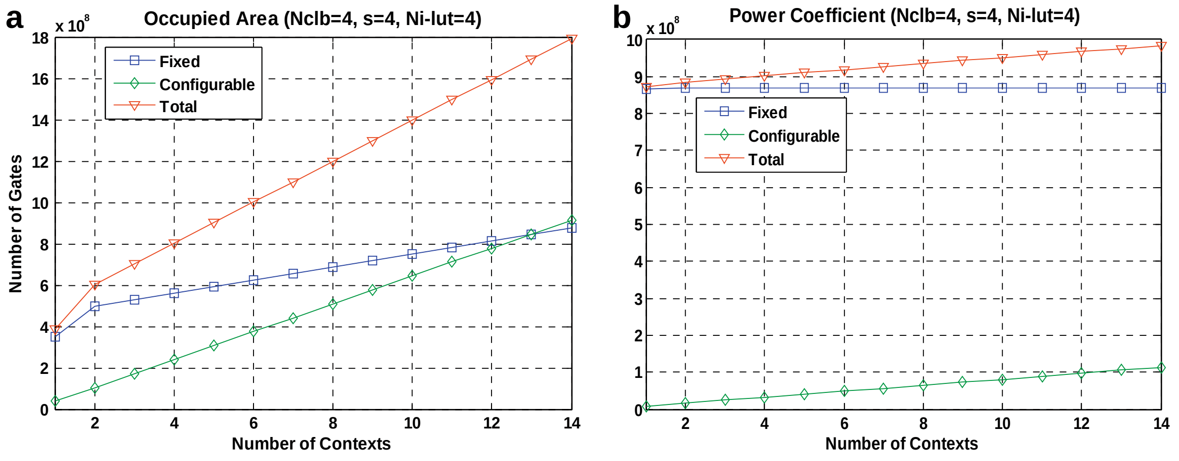


Fig. 8. CMOS MC-FPGA area (a) and power (b).

An evaluation of the local storage FPGAs using CMOS cells immediately reveals that they have two main disadvantages: the consumed power and the area occupied by the standby cells. Therefore, the challenge is to reduce these two factors. Our proposed solution, which is based on a non-volatile memory cell (NROM) [3], is described in the next section.

5.2. NROM based local storage

The NROM cell operates very similarly to flash cells; however, it can store more than one bit. Current technology enables to store two, four and even eight bits in a single cell; however, only two bit cells are used here, as otherwise very complex reading circuitry is required. NROM cells can only be rewritten a limited number of times and have long access time. Therefore, they cannot be used for active configuration cells. However, the NROM cells are non-volatile and can be switched off, so they can be used efficiently to store the standby configurations, which are assumed to change infrequently. The proposed hybrid cell uses a fast CMOS cell for the active configuration and efficient NROM cells to store (standby) copies of all "local" configurations, be they active or on standby. By so doing, these NROM cells are only read during local configuration changes.

5.2.1. NROM based multi-context memory block

NROM cells normally require complex access circuitry, which constitutes unacceptable overhead when applied to a small number of cells. Instead, we propose a memory block (Fig. 9) that is very efficient and has a small number of control transistors. The proposed memory block has K inputs that select the desired memory cell by EN_i, along with SEL_0, SEL_1 signals that choose one of the two bits using a reverse voltage reading technique [3]. NROM cells M_1 thought M_K hold the data bits, and transistor pairs (T_3, T_4) and (T_5, T_6) are responsible for the control over the drain-source voltage polarity of the NROM cells, thereby determining which of the cell's two bits are accessed. The MXVAND [20] simulation model was used for simulation of the second bit's effect.

The memory block operates using a dynamic circuit technique, which employs 'Domino' technology. The operation cycle is divided into two phases as shown in Fig. 10.

In the first phase (CTRL = 0), the output (C_{load}) is pre-charged and a desired cell is chosen by the combination of the decoded (EN_i) signals and selection (SEL_i) signals; next, the CTRL signal goes high and the result is presented at the cells' output (C_{load}).

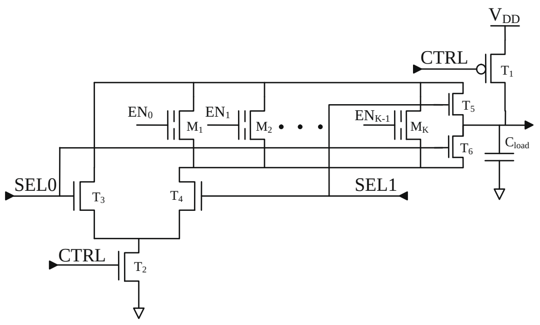


Fig. 9. Memory block based on NROM dual gate cell.

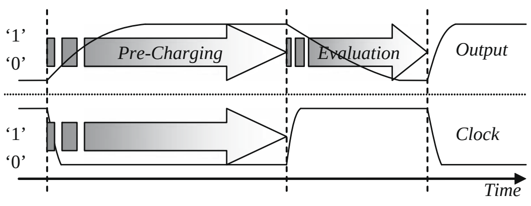


Fig. 10. Dual-bit memory cell operation model.

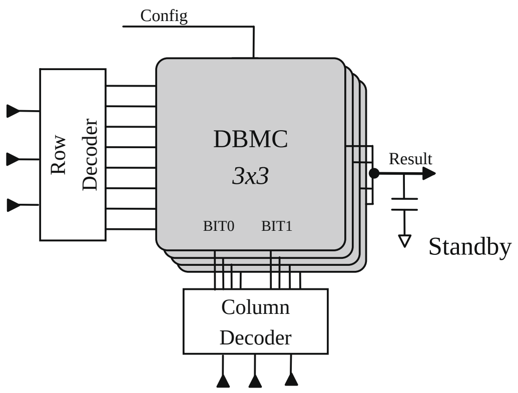


Fig. 11. Memory block based on dual bit memory cell.

The use of NROM cells decreases the actual number of decoders' input lines to $N-1$, thereby halving the decoder area. However, for a decoder with a large number of input lines, the area and the latency are still very expensive. In order to overcome these problems, we propose a "3D" Row/Column access architecture that reduces the decoder depth, thereby decreasing both its area and access time. Fig. 11 depicts the row/column access architecture for a 64 bit memory block.

5.2.2. Area and power evaluation

The NROM based MC-FPGA architecture was evaluated under the same conditions as the CMOS based one presented in Section 5.1.

The occupied area grows "almost" linearly with an increase in the number of contexts. The "staircase" behavior reflects the fact that NROM cells can store two bits, so a new NROM cell is required only for an odd number of contexts. The NROM based architecture is more area efficient than its CMOS counterpart. For example, for $N_c = 8$ the difference in the total FPGA area is almost a factor of three, with areas of 4.2×10^8 (Fig. 12) and 12×10^8 (Fig. 8a) for NROM and CMOS based implementations, respectively.

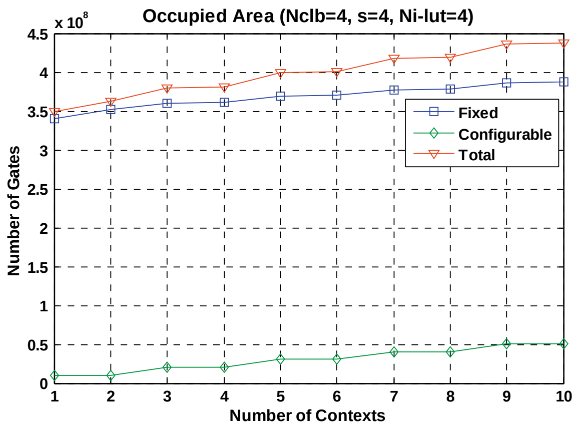


Fig. 12. NROM based MC-FPGA area.

The static power consumption is minimized by the use of the non-volatile NROM, which is switched off during operation and does not cause additional leakage currents. Therefore, the NROM based implementation is also preferable from the static power perspective.

5.3. Summary

The NROM based solution is preferable over the CMOS one in terms of both the additional area and the static power. While the local storage approach features fast reconfiguration, it raises a number of critical points:

- *Area efficiency* – replicated logic (selectors/decoders/Tri-State buffers) required for the control over the configuration changes.
- *Area flexibility* – whenever several instances of the same module should be active at the same time or should be allocated at different locations, this module would have to be placed in several configurable areas as required by the application. This increases cost and/or reduces flexibility.

The obvious alternative is to use a single, central repository for additional configuration data (contexts). However, that raises new problems, most notably:

- *Reconfiguration time* – the time required for a context switch is very long, reducing the efficiency of the multi-context device.
- *Communication "Hot Spots"* – whenever more than one module requires a context switch, there is a communication contention among these operations.

The global storage approach is thus reminiscent of the currently available partially reconfigurable devices [6]. We now present a hybrid architecture that addresses all the above points.

6. Hybrid storage multi-context FPGA

6.1. Architecture description

This section presents a new, Hybrid MC-FPGA architecture that combines central and local memories for storing configuration data, as depicted in Fig. 13. The HMC-FPGA is divided into configuration regions. Each region contains configurable logic clusters and memory cells for its own active configuration. It also stores a single standby configuration, which may be swapped in immediately upon request. Additionally, there is a central repository of all possible configurations. The input and output data streams of logical clusters are buffered in central memory modules, but can also be connected directly between logical clusters when they operate in synchrony.

6.1.1. Local storage

The local storage can be viewed as "double buffering," permitting the staging of the next active configuration in the background by copying it from the (slow) central repository to the local standby configuration slot, and then swapping it in instantaneously so as to not waste execution time. The content of both active and (local) standby configurations is thus expected to change numerous times during the lifetime of a chip, ruling out the use of NROM cells for storing them. We therefore use CMOS cells for both, with lower-power versions (slower) used for the standby configuration. Finally, note that this technique both hides the latency of copying from the central repository and "smoothes" the communication load on it, as multiple concurrent requests could otherwise cause temporary congestion and delays. As will be discussed shortly, it is also possible two regions to swap their standby configurations.

6.1.2. Central storage

The central configuration repository is not updated frequently, so NROM cells provide an area- and power-efficient solution. In those cases that do require frequent updates, a good alternative is LPDRAM [21] cells, which provide a dense and low static power solution.

Additional global space is allocated for intermediate state data such as states of flip flops: the data from an active module that is being deactivated is transferred into the global space. This data will be transferred back toward the appropriate functional module's reactivation (in the same or different location!). It is important to notice that the number of the intermediate buffers should be greater than the number of configurable modules in order to support multiple instances of the same module.

6.1.3. The reconfiguration mechanism

In order to perform the reconfiguration, sequential data transmission is used, similar to the well known JTAG [22] protocol. Separate lines are used for the configuration data and for flip–flop state transfers. Referring to Fig. 13, the configuration

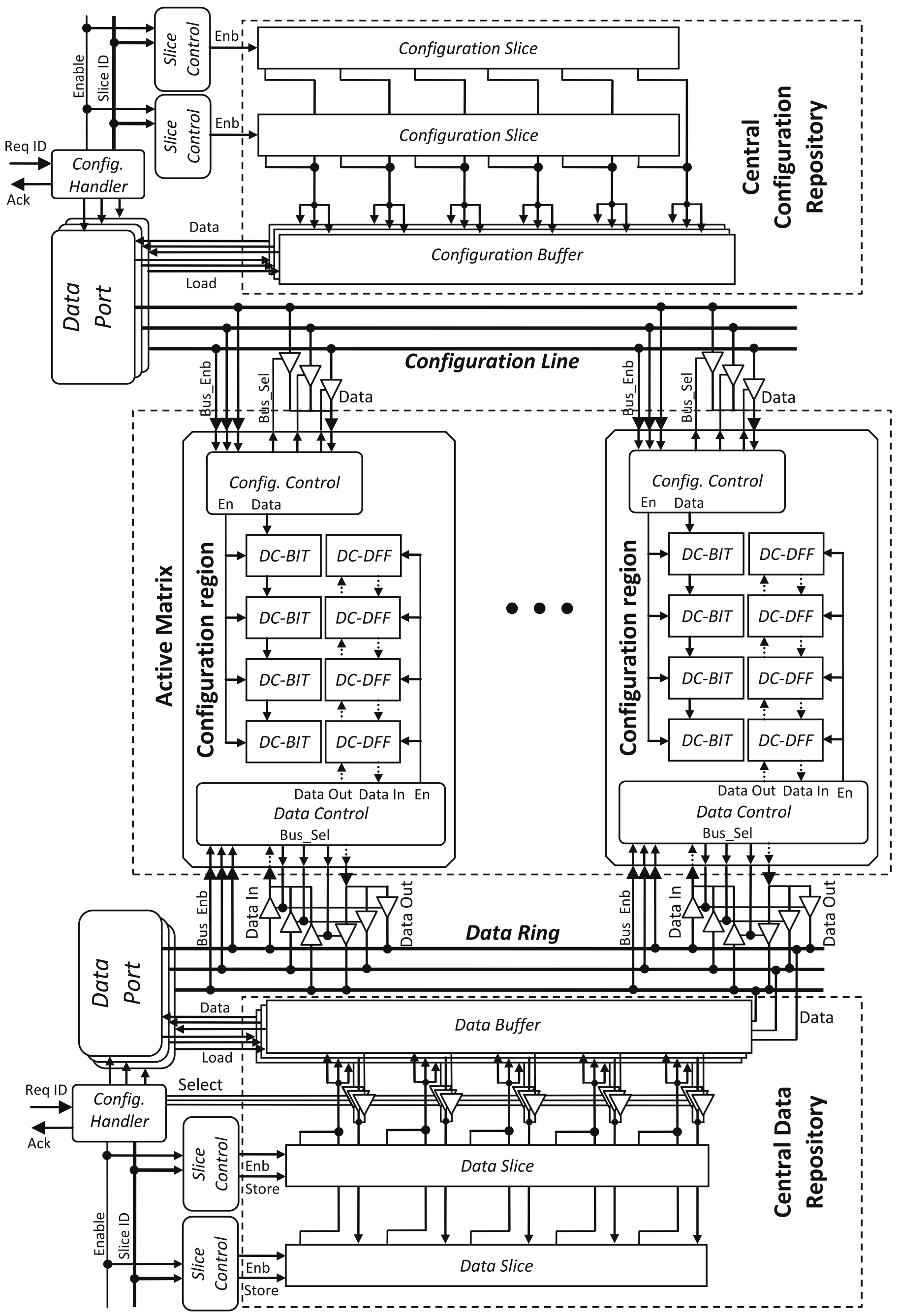


Fig. 13. A hybrid multi-context FPGA architecture.

data is transferred through the ‘Configuration Lines’. This transfer is unidirectional since the central configuration repository is not updated during reconfiguration. However, the flip–flop state transfers are bi-directional, since the current data should be saved for future use. The circular data path ‘Data Ring’ is used for these transfers. The reconfiguration speed depends on the width of the configuration busses. Fig. 13 depicts the general architecture of the proposed FPGA architecture.

The “JTAG” approach can be used to swap the active and standby configurations of a particular cell without requiring additional memory (Fig. 14). We furthermore propose its use for swapping the standby configurations of two different regions, which can relieve the communication load on the central repository. Of course, multiple swaps can be carried out concurrently without involving the central configuration resources. This opens interesting scheduling opportunities, and will be elaborated upon in Section 7.

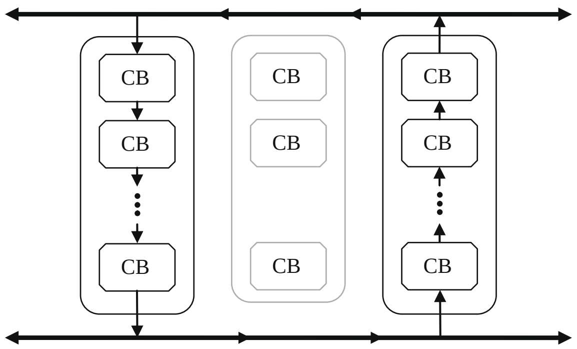


Fig. 14. Dual-context configuration bit circuit.

6.2. *Area and power evaluation*

The area and power models are extensions of previously presented models. The detailed models are presented in [23], with highlights in Appendix A.3. The baseline configuration for the evaluation has an Active Matrix similar to the CMOS based dual-context FPGA described in Section 5.1, a Configuration Repository of the size required to contain a single configuration of the active matrix, and a Data Repository of the size required to contain a single state of flip–flops located in the active matrix.

The effect on area and power is evaluated versus the size of configuration and data repositories, which is represented by a growth factor. The growth factor only affects the configuration and data repositories; the Active Matrix remains constant. This method is analogous to the method described in Section 5.1, wherein the number of contexts (N_C) was increased. However, here the additional standby configurations and intermediate data are added externally to the Active Matrix, as only one standby configuration is stored locally at each region. Fig. 15 depicts the results of area (a) and power (b) evaluations of the architecture, which uses LPDRAM [21] cells for both configuration and data storage; note that the ordinate is logarithmic.

The evaluation shows that in the baseline configuration (growth factor = 1), the Active Matrix accounts for over 98% of the total area and of the static power consumption. As the amount of configuration data is increased, both the area and the power coefficient grow linearly. For a growth factor of 8, the additional area occupied jointly by configuration and flip–flop states (data) (the sum of the respective curves) is still negligible, ~7% of the total FPGA area (Fig. 15a). However, for this growth factor the static power consumed by the configuration repository grows up to 20% of the total static power (Fig. 15b). The static power consumed by the data repository may be neglected, constituting less than 0.5% of the total power. Significant improvements in the static power consumption can be achieved by using NROM [3] cells, which are active only during the reconfiguration process. This can be seen in Fig. 16, which depicts the results for area (a) and static power (b) evaluations for the NROM based configuration repository.

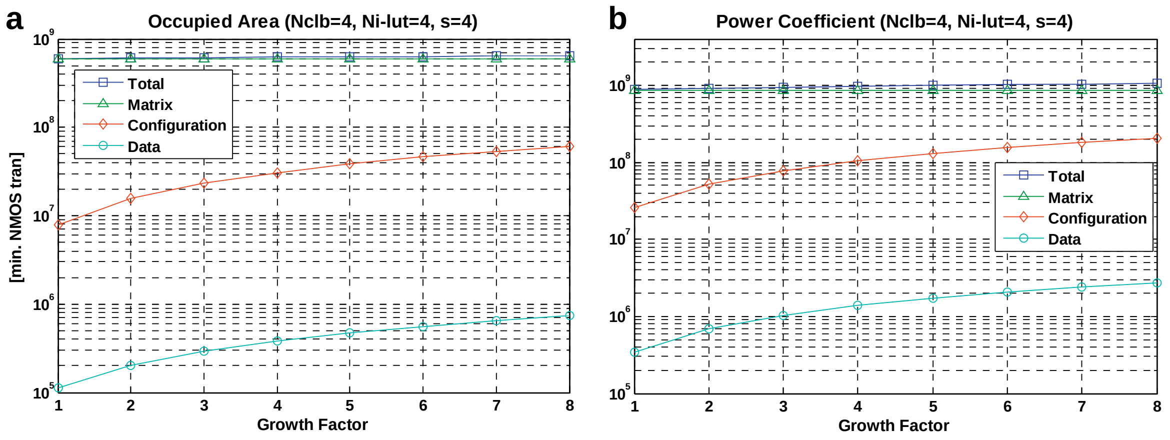


Fig. 15. Hybrid LPDRAM MC-FPGA area (a) and power (b) analysis.

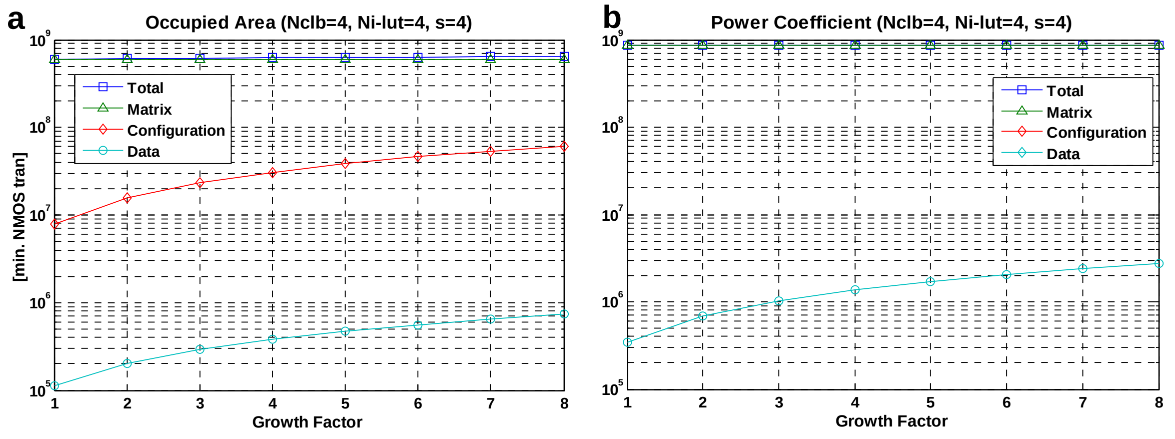


Fig. 16. Hybrid NROM MC-FPGA area (a) and power (b) analysis.

7. Multi-context FPGA comparison and evaluation

This section summarizes the comparative architecture evaluation in terms of required area and static power consumption, and provides guidelines that help in determining the preferred architecture for a specific application.

In order to perform the comparison between single context and multi-context approaches, the single context FPGA baseline is also extended by the growth factor (replicating the active part). With this, the potential number of configurations is equal among all architectures, but the number of active configurations is larger for the baseline. Such a comparison reveals the area and power savings of the MC-FPGA when there is no need for all configurations to be active concurrently, so the comparison is very meaningful even if not "fair" to the baseline architecture.

7.1. Area and power summary

7.1.1. Area

As depicted in Fig. 17a, the Hybrid architecture enables efficient data storage. For a growth factor of two and above, the total area of the Hybrid Architecture is smaller than the total area of the Local Storage Architecture based on the CMOS cells. This is related to the fact that the Hybrid Architecture is based on the Dual-Context Active Matrix that can be interpreted as a local storage multi-context FPGA with doubled configuration data capacity. When comparing the local storage architectures, there are measurable differences in occupied area between CMOS and NROM approaches. This is due to NROM cell area density, as each NROM cell occupies less area and can store two configuration bits. For the Hybrid architecture, the total area is hardly influenced by the memory cell technology when NROM or LPDRAM are used.

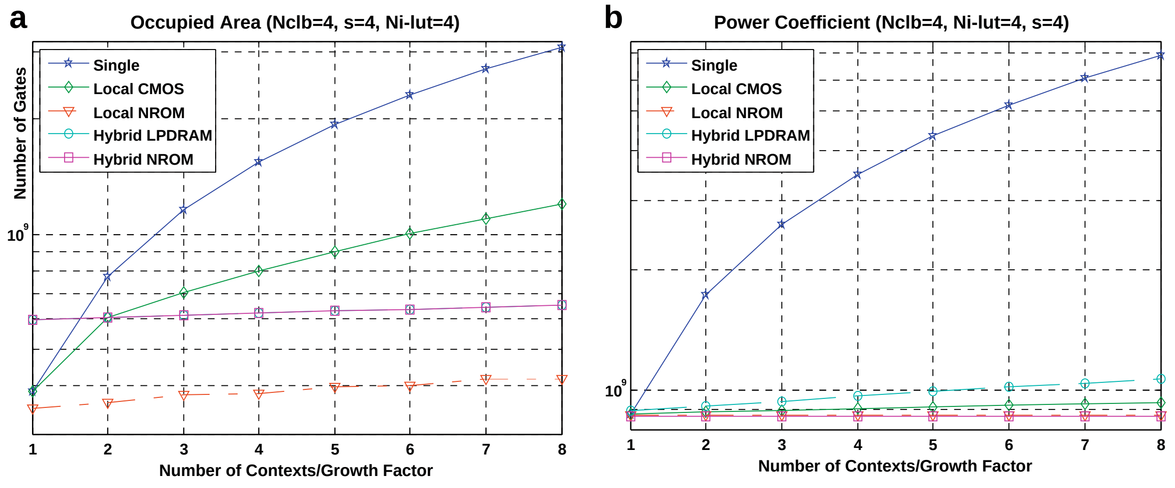


Fig. 17. FPGA architectures-area (a) and power (b) comparison.

Let us return to the example of the MD5 algorithm described in Section 1 (Fig. 1). Since only one of the four blocks can be used at any given time, the relevant growth factor is 4. In this case the local storage, CMOS based implementation occupies an area equivalent to 50% of the single context device; both hybrid architectures are 40%, and the local storage NROM based device is only 25%.

7.1.2. Power

For the static power comparison, a static power coefficient that takes into account the power supply voltage and the circuit design parameter was defined. With a single context, over 98% of the static power is consumed by the Active Matrix. Therefore, there are no significant differences in static power between the Local and Hybrid architectures. However, when increasing the total number of configurations one can observe (Fig. 17b) an increase in the static power consumed by volatile memory architectures (CMOS and LPDRAM). It is obvious that the NROM based architectures are more power efficient, because they do not require power during standby. While the multi-context approach is superior, there is no significant difference between the various multi-context architectures. For the MD5 example, the average static power savings relative to the single context baseline is 25%.

7.2. Application mapping guidelines

Since the two previously described architectures have different behavior, there is no single best architecture for all applications. The following guidelines should help in matching an appropriate MC-FPGA architecture to an application.

Local storage MC-FPGA. Matching application characteristics:
- A simple scheduling scheme and predefined execution.
- Simple relations among executed modules.
- Short execution periods (thus requiring fast context switching).

This architecture is thus mostly suitable for small and simple applications.

Hybrid-storage architecture. This, in contrast, is well suited to complex and large applications that have the following characteristics:
- Complex/Unknown Execution Scheme.
- Long execution cycle.
- Multi-Threading execution environment.

Good examples of such applications are video processing and codec applications or even complex embedded systems like cellular phones or other mobile devices.

Having established the viability of dynamic configuration, we next turn our attention to some higher level scheduling issues that are closely tied to our proposed Hybrid architecture.

8. Swap-based configuration pre-fetching

The execution schedule of an application can be viewed as a (time, space) assignment of configurations to the active slots of regions. In this section, we assume that a partial schedule is provided in the form of the time intervals during which any given configuration (function) should be active, and that all configuration regions are identical. It thus remains to decide the mapping of active configurations to regions and to schedule the pre-fetching of the next active configuration for each region. Our focus is on the Hybrid architecture.

Unlike the central storage architecture, in which next configurations are fetched from the central repository, the Hybrid and Local configurations (also) permit the swapping of standby configurations between locations. This permits exploitation of the inherent parallelism in interconnection hardware, thereby avoiding a potential communication bottleneck at the central repository. However, it is not clear whether and how this can be exploited. For example, why would swapping configurations between two locations, both of which are capable of executing either one, be beneficial? In the remainder of this section, we provide some insights into the viability of swap-scheduling, leaving actual scheduling for future research.

8.1. The model

There are N execution locations, each with a single active configuration and M standby configurations that are stored locally (Fig. 18). The existence of a central repository is irrelevant for this discussion. There is a set of tasks (functions), each of which requires a single configuration slot at some location.

We consider a set of applications, each of which requires the (serial) execution of a sequence of tasks, corresponding to a linear precedence graph containing one or more of the tasks (Fig. 19).

We assume a discrete time axis, and that the execution of any given task as part of any given application requires an integer number of time slots. The execution time of a given task for different applications may differ, for instance due to different amounts of data, but is assumed to be known in advance. The execution of an application may migrate from one location to another instantaneously on task boundaries. (This makes sense when task output and inputs are stored in a central memory.)

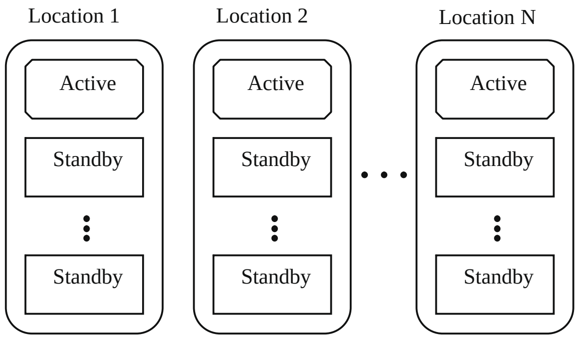


Fig. 18. Multi-context FPGA model.

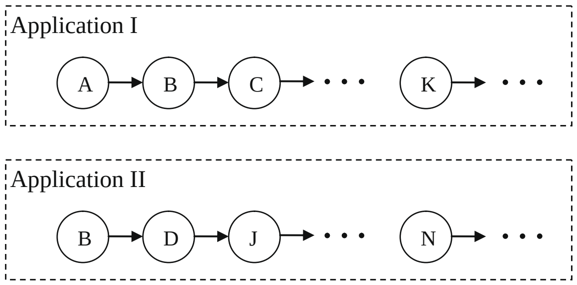


Fig. 19. Sample application execution graph.

We assume a full system, i.e., there is room for only one copy of each function in the entire system, so at most one instance of any given function can be active at any given time. Also, at most one task of any application may be active in any given time slot. Lastly, we assume that an inter-location configuration swap takes less than a single time slot and an active/standby swap in the same location is instantaneous.

8.2. The problem

Given a set of applications and locations, can the configuration swap mechanism be used to prevent a stall? (This is an offline time/space scheduling issue, and even one example of stall prevention is considered success.)

Remark. Active tasks cannot be swapped during execution, so an inter-location swap involving an active configuration would waste a time slot. Consequently, the only relevant inter-location swaps are between standby configurations of two locations.

Lemma 1. *In a system with a single standby configuration per location and tasks of equal duration (one time slot without loss of generality), the Configuration Swap method is useless.*

Proof. Consider two 2-task applications and two locations. To enable the completion of both applications in two time slots, they must execute different tasks in each of those slots. Let us denote the tasks executed by the two applications in the first slot by A and B, respectively. These tasks are obviously the initially active tasks of the respective locations. We denote the respective standby tasks C and D. In the second time slot, each application may wish to execute any task other than the one it executed in the first slot, and these tasks must be different. Consider the exhaustive set of legal combinations of 2nd-slot tasks:

- (B,A): The original two tasks remain active and the applications migrate.
- (C,D): A local active/standby swap at each of the locations.
- (B,C) or (D,A): Both applications migrate, and a local active/standby swap occurs at one location.
- (B,D) or (C,A): C and D should be swapped at the outset. Migration of both applications and a local swap at one location would solve the problem.
- (D,C): C and D should be swapped at the outset, and two local swaps would solve the problem.

Since the above are the only legal combinations that do not mandate a stall, swapping standby configurations is not needed for preventing preventable stalls). □

Lemma 2. *In a system with a single standby slot per location and variable task durations, an inter-location configuration swap may prevent a stall.*

Proof. Consider the following two-application example (Fig. 20). The tasks executed by the different applications appear with different backgrounds in order to enable tracking despite application migration. In each location and for each time slot, the active task is shown along with the standby one in parentheses. An inter-location swap of the standby tasks is denoted by diagonal arrows.

In order to prevent stalls, concurrently running tasks must reside in different locations (when active). Accordingly, the tasks in each of the pairs (A,B), (C,D), (B,D), (B,C) must reside in different locations. This, however, is impossible, since B, C and D must reside in three different locations yet there are only two. So, static placement, even when accompanied by instantaneous application migration on task boundaries, does not help. Fig. 20 depicts a possible solution employing an inter-location swap. During the 3rd time slot, the standby configurations (A and C) are swapped in preparation of C's being active concurrently with B in the following time slot, preventing the stall. □

Lemma 3. *In a system with multiple standby configurations per location, an inter-location configuration swap may prevent a stall even if every application changes tasks every time slot.*

Proof. By example. Consider a dual location system with two standby configuration slots per location, and the two applications depicted in the top part of Fig. 21.

Let us attempt to statically partition the tasks between the two locations (Fig. 21). After arbitrarily placing task A in location 1, the simultaneous execution requirements dictate that D and E must be in 2, and consequently that B and C must be in 1. The simultaneous execution of D and F requires F to be at location 1, but that location is already full, so static placement fails. The bottom part of Fig. 21 shows how a swap of standby configurations D and F during the 4th time slot solves the problem, completing the proof. □

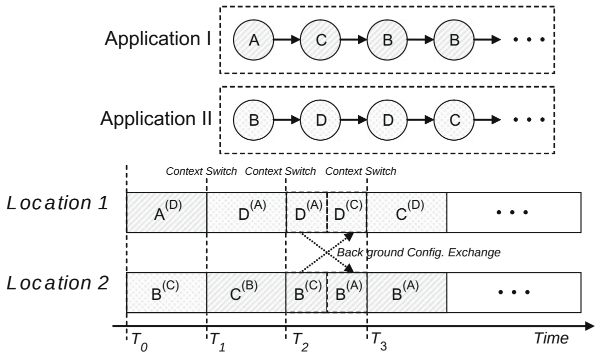


Fig. 20. Single standby location example.

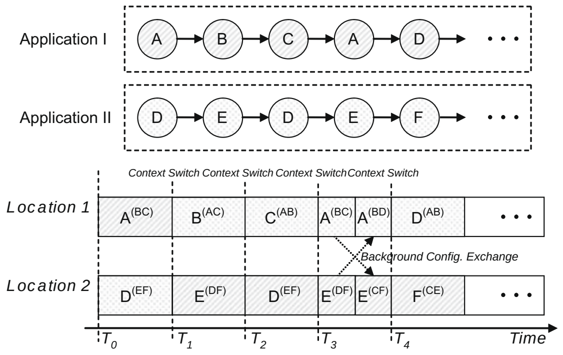


Fig. 21. Dual standby location example.

9. Conclusions and directions for further research

Incrementally-reconfigurable FPGAs can reduce the amount of hardware (and related power consumption and cost) required for computing at any given rate. In HPC applications wherein high degrees of data parallelism are available, one can instead achieve much higher degrees of parallelism with the same hardware and static power by (also) loading multiple instances of the same function at any given time rather than a fixed configuration comprising different functions that are not needed concurrently.

The main contributions of this work are at the circuit, logic design and architecture levels, as well as some important scheduling insights. An efficient circuit for access to small flash based memory units that normally require complex access circuitry was proposed. Also, two viable and efficient architectures for multi-context FPGAs were presented: one is based on distributed configuration data storage, wherein the additional configurations are stored beside the active logic; the other is based on a hybrid storage approach wherein only a single "standby" configuration is stored beside the active logic while the rest of the configuration data is stored outside the active area. It was shown that even for local storage architectures, non-volatile memory may be used efficiently. Problems caused by local storage architectures were identified, and were solved by a new hybrid architecture that overcomes area assignment and slow reconfiguration problems. This architecture gave rise to configuration-swap opportunities that alleviate possible communication bottlenecks with a central configuration repository, and we presented some preliminary insights pertaining to the usefulness of such swaps.

This work is an important step toward hybrid multi-context architectures, and opens opportunities for further research at both the architecture and algorithmic (scheduling) levels. Architectural issues include the determination of the optimal number of contexts stored locally and investigation of combinations of volatile and non-volatile configuration memory inside the Active Matrix (MC-Cell).

This work only addressed the hardware platform, which must be supported by appropriate scheduling algorithms and execution environments. Topics for further research include task execution scheduling, configuration pre-fetching (avoidance of communication "hot spots," hot configuration switching and exploiting concurrent "local to local" circular swaps), and multi-core/multi-tasking execution environments.

Appendix A. FPGA model derivation – example

This section demonstrates how the area and power models were derived for the various FPGA architectures covered in this paper, using the single context FPGA model as an example. Models for the remaining architectures were derived similarly with small adaptations. The detailed model derivation can be found in Section 6 of [23].

A.1. Single context FPGA model

A.1.1. FPGA model parameters

This section presents a set of parameters used for the FPGA area and power evaluation. The number of clock trees is defined by N_{clock}, The total number of configurable logic clusters (CLC) in an FPGA is denoted by N_{cluster}. The number of configurable logic blocks (CLB) in a cluster is denoted by N_{CLB}. $N_{\text{i-LUT}}$ denotes the number of inputs to the Lookup tables. The routing wires' length is given in units of the Manhattan length and denoted by s. The routing segment, the area between two configurable routers, consists of a number of wires, denoted by W. Table A1 summarizes the FPGA model parameters.

A.1.2. Address decoder

The Lyon–Schediwy decoder [A1] is used for address decoding within memory cells. This decoder reduces the occupied area by reducing the number of bulky PMOS transistors, and increases decoding speed by sharing pull-up transistors for the NOR gates. Fig. A1 depicts schematics of a 3 → 8 Lyon–Schediwy decoder.

Let N be the desired number of address lines; the number of the decoded lines will be 2^N. Each decoded line requires N NMOS transistors, so the area occupied by the NMOS transistors is $A_{\text{NMOS}}(N) = N \cdot 2^N$. The PMOS transistors have a tree-like structure with a height of N. The PMOS transistor width grows by factor of two for each tree level. In order to satisfy symmetric rise/fall timing, the initial width of PMOS transistors must be 3.5λ. The total PMOS area is therefore given by:

Table A1
Single context FPGA model parameters.

Symbol	Description
N_{clock}	Number of clock trees
$N_{\text{i-LUT}}$	Number of inputs to a LUT
N_{CLB}	Number of CLBs in a logic cluster
$N_{\text{i-cluster}}$	Number of external inputs to the logic cluster
s	Manhattan length of a routing wire
W	Number of wires in a routing segment
N_{Cluster}	Number of clusters in the FPGA

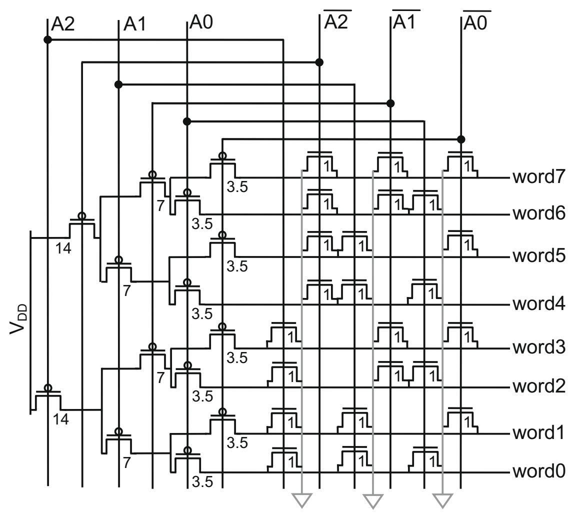


Fig. A1. 3 → 8 Lyon-schediwy decoder.

$$A_{\mathrm{PMOS}}(N) = \sum_{i=0}^{N-1} 3.5 \times 2^i \times 2^{N-i} = \sum_{i=0}^{N-1} 3.5 \times 2^i \times 2^N \times 2^{-i}, \tag{A1}$$

$$A_{\mathrm{PMOS}}(N) = \sum_{i=0}^{N-1} 3.5 \times 2^N = 3.5 \cdot 2^N \cdot N.$$

Therefore, an approximate decoder area model is given by:

$$A_{\mathrm{decoder}}(N) = A_{\mathrm{NMOS}}(N) + A_{\mathrm{PMOS}}(N) = 4.5.2^N \cdot (N). \tag{A2}$$

As previously mentioned, the address decoder can be represented as a set of *N-input* NOR gates. The *pull-up* transistors are serially connected, so their contribution to k_{design} diminishes with increasing stacking level. In contrast, the *pull-down* transistors are the major contributors to k_{design}. During decoder operation, only one address line is excited while the others remain low. The excited line causes leakage of the *pull-down* transistors, and the inactive lines cause the *pull-up* transistors to leak.

The total number of *pull-down* transistors in a single excited line is n, and each transistor contributes $\hat{K}_{\mathrm{transistor}} = 11/2 = 5.5$ to the power coefficient (according to the static logic coefficient given in the Table 2). We assume that all *pull-down* lines have similar behavior, so they are treated identically. Eq. (A3) expresses the power coefficient of the single excited line:

$$\hat{K}_{pull-down}(n) = n \times 5.5. \tag{A3}$$

The *pull-up* lines have different behavior, which depends on the stacking factor. The stacking factor varies from 1 up to the number of input lines. Table A2 presents leakage savings ratio as was described in [A2]. We define the discrete function $Sr(k)$ that models values in this table.

The total leakage current is calculated by accumulation of all pull-up lines. It can be shown that the number of lines with stacking factor k is equal to C_n^k. Thus, the power coefficient of the pull-up lines is:

$$\hat{K}_{pull-up}(n) = \sum_{i=1}^{n} (C_i^n \times Sr(i) \times 5.5). \tag{A4}$$

The final power coefficient $\hat{K}_{decoder}(n)$ of the address decoder with n address lines is:

$$\hat{K}_{decoder}(n) = \hat{K}_{pull-up} + \hat{K}_{pull-down} = 5.5 \times (n + \sum_{i=1}^{n} (C_i^n \times Sr(i))). \tag{A5}$$

Table A2
Leakage savings versus stack height.

Transistors in stack – k	1	2	3	4	5	6
Savings ratio – $Sr(k)$	1	9	16	20	24	27

A.1.3. Memory cell – D-FF

The role of the *D-FF* is to store temporary data of the *LUT* results. A common *D-FF* consists of two sequential latches that operate on opposite clock edges. Fig. A2 depicts the *D-FF* structure. Additional buffers are required in order to overcome races between Master and Slave latch signals. The proposed *D-FF* implementation has *12 NMOS* and *8 PMOS* transistors. The total *D-FF* area is thus (see Fig. A2):

$$A_{D-FF} = A_{\mathrm{NMOS}} + A_{\mathrm{PMOS}} = 12 \times 1 + 8 \times 3.5 = 40. \tag{A6}$$

The power coefficient of the *D-FF* cell is based on a *D-Latch* model expressed in Table A2. Eq. (A7) can be used to relate the k_{design} and *N* parameters:

$$\hat{K}_{D-FF} = 2 \times k_{D-Latch} \times N_{D-Latch} = 2 \times 10 \times 2.0 = 40. \tag{A7}$$

A.1.4. Memory cell – the configuration bit

The role of the *Configuration Bit* unit is to store the FPGA configuration data. The *Configuration Bit* consists of two sequential latches that operate on opposite clock edges, as depicted in Fig. A3. During operation only a one bit is active, and another bit is used only for the reconfiguration process. Due to the large number of reconfiguration items, it is impossible to use an address decoder for the configuration access. Therefore, the configuration process is performed serially and an additional latch is required in order to support the shift operation.

The total area required by a single configuration Bit is

$$A_{CB} = 2 \times A_{CMOS} + 5 \times A_{NOT} = 22 + 22.5 = 44.5. \tag{A8}$$

As described previously, the above circuit has two modes of operation. We will ignore the reconfiguration mode because it is active very seldom and thus does not affect long term static power consumption. Therefore, the power coefficient is based on the 6-transistor *CMOS* memory cell model presented in [13] and is given by Eq. (A9):

$$\hat{K}_{CB} = K_{6T} \times N_{6T} \times \left(1 + = \frac{1}{e}\right) = 4.92. \tag{A9}$$

A.1.5. Multi-stage driving buffer

The multi-stage buffers are used to drive long wires that have large capacitance and are usually used at the output stages. These buffers are fast, but require more area. In order to achieve the minimum propagation delay for the whole chain, the input capacitance of every stage should grow (approximately) by a constant factor. Thus the effective resistance and the load

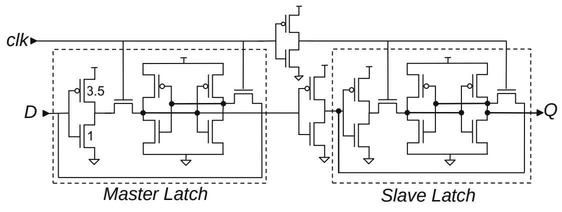


Fig. A2. Single content D-FF layout.

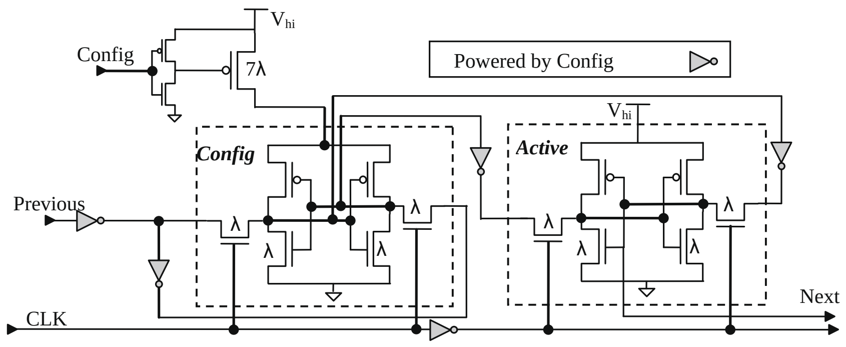


Fig. A3. A single content configuration bit layout.

capacitance of every stage are growing with the same factor, yielding an equal propagation time for every stage. The optimal number of stages (N) and stage growth factor (A) are presented in [A3] and expressed by Eqs. (A10) and (A11):

$$N = \left\lceil \ln\left(\frac{C_L}{C_{in}}\right) \right\rceil \tag{A10}$$

$$A = \sqrt[N]{\frac{C_L}{C_{in}}}. \tag{A11}$$

where C_L is a load capacitance and C_in is the first stage input capacitance. When the number of stages is predefined, the stage growth factor can be derived as follows:

$$\frac{C_\text{L}}{C_\text{in}} = \exp\{N\}.$$

$$A = \sqrt[N]{\frac{C_\text{L}}{C_\text{in}}}\sqrt[N]{\exp\{N\}} = \exp\{1\} = e. \tag{A12}$$

Fig. A4 depicts the structure of the multi-stage driving buffer with driving strength f.

The total area of the driving buffer is:

$$A_{buffer}(f) = A_{NOT} \times \sum_{i=0}^{f-1} e^i = (1+3.5) \times \sum_{i=0}^{f-1} e^i = 4.5 \times \frac{e^f - 1}{e-1}. \tag{A13}$$

The derived power coefficient $\hat{K}_{buffer}$ is based on the *static logic* k_{design} parameter as shown in Table A2. The power coefficient of the first stage *NOT* gate is defined as $\hat{K}_{NOT} = N_{NOT} \times k_{static} = 2 \times 11 = 22$. However, we will extend this parameter considering the *buffer* width growth. Eq. (A14) expresses the multi-stage driving buffer power coefficient.

$$\hat{K}_{buffer}(f) = \hat{K}_{NOT} \times \sum_{i=0}^{f-1} e^i = 22 \times \frac{e^f - 1}{e-1}. \tag{A14}$$

A.1.6. Power gated buffer

The use of *power gated buffers* (*PGB*) will result in static power reduction. The buffer is built of multi-stage driving buffer (as depicted in Fig. A4), the configuration cell that holds the buffer state, and the control logic (*NOR* and *NAND* gates) that prevents operation during the reconfiguration process. Structure of power gated buffer is depicted in Fig. A5.

The power gating transistors T_1, T_2 must be strong enough to support the buffer logic. Thus their width should be at least as large as that of the last buffer stage, and hence should be multiplied by factor e^{s-1}. The area of the *PGB* is therefore:

$$A_{PGB}(s) = A_{CB} + A_{power} + A_{buffer} + A_{control} = 67 + 1.22 \times A_{buffer}(s). \tag{A15}$$

The power coefficient of the *PGB* is given by the composition of its components' power coefficients. The PGB may operate in two different modes: in *enabled* mode, the power supply transistors are open and therefore the leakage current depends on the parameters of the driving buffer; in *disabled* mode, the power transistors are shut down and therefore the leakage current is significantly reduced. The additional leakage factors are the *configuration bit* and *control logic*. Eq. (A16) expresses the static power coefficients for the two modes of operation:

$$\begin{aligned} \hat{K}_{PGB}^{active}(s) &= \hat{K}_{buffer}(s) + \hat{K}_{CB} + \hat{K}_{control} = 22 \times s \times \frac{e^s - 1}{e-1} + 29.4, \\ \hat{K}_{PGB}^{s\tan dby}(s) &= \frac{\hat{K}_{buffer}(s)}{e} + \hat{K}_{CB} + \hat{K}_{control} = 22 \times \frac{s}{e} \times \frac{e^s - 1}{e-1} + 29.4. \end{aligned} \tag{A16}$$

A.1.7. Connection point

The role of the connection point is to output the logic result from the *CLB* to the routing wires. Unlike the *Power Gated Buffer*, the *Connection Point* doesn't strengthen the output signal. It is connecting external routing lines as depicted in

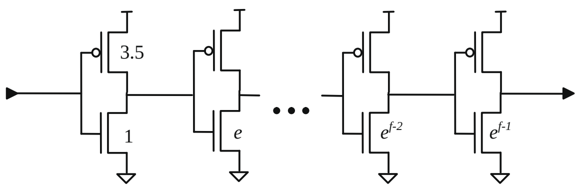


Fig. A4. A multi-stage driving buffer structure.

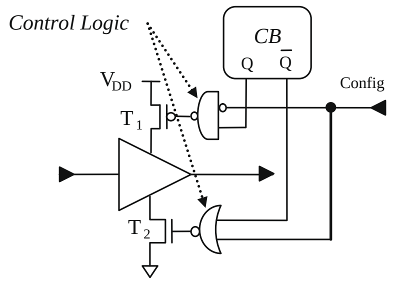


Fig. A5. Configurable power gated buffer structure.

Fig. 4. It comprises a CMOS pair of path transistors, and is controlled by a single memory cell. The control logic (*NOR* and *NAND* gates) prevents operation during the reconfiguration process. Fig. A6 depicts the *Connection Point* structure.

The area consumed by the control gates is 22.5 minimum width *NMOS* transistors. The area occupied by the pass transistor pair depends on strength factor '*f*', which affects pass transistor width. The width multiplier is given by e^{f-1}. The total area of the *Connection Point* is:

$$A_{CP}(f) = A_{control} + (1 + 3.5) \times e^{(f-1)} + A_{CB} = 67 + 4.5 \times e^{(f-1)}. \tag{A17}$$

The power coefficient of the *connection point* combines the power coefficients of the *configuration bit* and the *control logic*. This model is very similar to the presented *PGB* power coefficient model. Eq. (A18) expresses the model parameter:

$$\hat{K}_{CP}(f) = \hat{K}_{CB} + \hat{K}_{control} = 7.2 + 22 = 29.2. \tag{A18}$$

A.1.8. Lookup table (LUT)

The central memory storage unit is the *LUT*. It holds the functional logic values. Each LUT consists of an address decoder and a memory matrix, which is based on *CMOS* memory cells. The LUT can also be presented as common *CMOS* memory – *SRAM*. Fig. A7 depicts the LUT structure, followed by the area estimation model.

$$A_{LUT}(N) = A_{decoder}(N) + 2^N \times A_{CB} = 4.5 \cdot 2^N \cdot N + 2^N \times 44.5A_{LUT}(N) = 2^N \times [4.5 \cdot N + 44.5]. \tag{A19}$$

The power coefficient of the *LUT* combines power coefficients of the *address decoder* and a group of *configuration bits*. The *LUT* power coefficient is given by

$$\hat{K}_{LUT}(n) = \hat{K}_{decoder} + 2^n \times \hat{K}_{CB} = (2^n - 1) \times \frac{5.5}{e^n} + 5.5 \times n + 44.2 \cdot 2^n. \tag{A20}$$

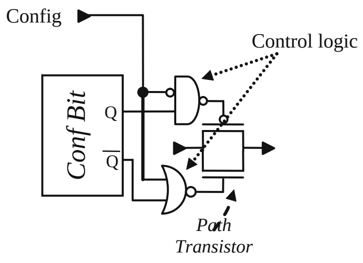


Fig. A6. Connection point structure.

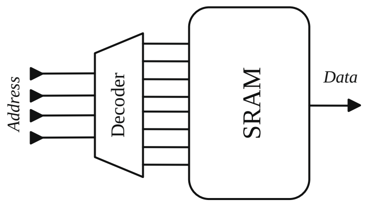


Fig. A7. A lookup table structure.

Table A3
CLB area model parameters.

Module	Amount
LUT	1
D-FF	1
SELECT ($2 \rightarrow 1$)	1
SELECT(($N_{i\text{-}Cluster}$+N_{CLB}) → 1)	$N_{i\text{-}LUT}$
Connection point	W
4× Buffer	1

A.1.9. Configurable logic block

The Configurable Logic Block (*CLB*) is a first level compound unit and is a component of a *Configurable Cluster* as depicted in Fig. 4. Table A3 summarizes the components used in each *CLB* unit and it is followed by occupied area estimation:

$$\begin{aligned} A_{CLB}(N_{CLB}, N_{i-Cluster}, N_{LUT}, W) &= A_{LUT}(N_{i-LUT}) + A_{D-FF} + N_{i-LUT} \times (A_{MUX}(N_{i-Cluster} + N_{CLB}) + \lceil \log_2(N_{i-Cluster} + N_{CLB}) \rceil \\ &\quad \times A_{CB}) + A_{MUX}(2) + A_{4x} + W \times A_{CP}(4) \\ &= A_{LUT}(N_{i-LUT}) + 40 + W \cdot (67 + 4.5 \cdot e^3) + N_{i-LUT} \times (A_{MUX}(N_{i-Cluster} + N_{CLB}) \\ &\quad + \lceil \log_2(N_{i-Cluster} + N_{CLB}) \rceil \times 44.5) + 16.7 \\ &= A_{LUT}(N_{i-LUT}) + N_{i-LUT} \times (A_{MUX}(N_{i-Cluster} + N_{CLB}) + \lceil \log_2(N_{i-Cluster} + N_{CLB}) \rceil \times 44.5) \\ &\quad + 157.4 \cdot W + 56.7. \end{aligned} \tag{A21}$$

The power coefficient of the *CLB* is a complex one a derived by composition of the *CLB* component factors and is represented in Eq. (A22):

$$\begin{aligned} \hat{K}_{CLB}(N_{CLB}, N_{i-Cluster}, N_{LUT}, W) &= \hat{K}_{LUT}(N_{i-LUT}) + \hat{K}_{D-FF} + N_{i-LUT} \times (\hat{K}_{MUX}(N_{i-Cluster} + N_{CLB}) + \lceil \log_2(N_{i-Cluster} + N_{CLB}) \rceil \times \hat{K}_{CB}) \\ &\quad + \hat{K}_{MUX}(2) + \hat{K}_{4x} + W \times \hat{K}_{CP}(4). \end{aligned} \tag{A22}$$

A.1.10. Configurable cluster

The next architecture level is the logic cluster that contains a number of *CLBs* as described in Section 4.1. The number of input multiplexers in the logic cluster is equal to the number of inputs. The number of inputs to each multiplexer is equal to the number of vertical routing lines. Finally, the number of output connection points equals the number of the *CLBs* in the cluster times the number of vertical routing lines. Eq. (A23) expresses the area of the logic cluster:

$$\begin{aligned} A_{Clsuter}(N_{CLB}, N_{i-Cluster}, N_{i-LUT}, s, N_{clock}, W) &= N_{CLB} \times A_{CLB}(N_{CLB}, N_{i-Cluster}, N_{i-LUT}) + N_{i-Cluster} \times (A_{MUX}(W) + \lceil \log_2 W \rceil \times A_{CB}) \\ &\quad + A_{MUX}(N_{clock}) + \lceil \log_2 N_{clock} \rceil \times A_{CB}. \end{aligned} \tag{A23}$$

Like the *CLB* power coefficient, the power coefficient of the configurable cluster is given by composition of its components and is expressed in Eq. (A24):

$$\begin{aligned} \hat{K}_{Clsuter}(N_{CLB}, N_{i-Cluster}, N_{i-LUT}, s, N_{clock}, W) &= N_{CLB} \times \hat{K}_{CLB}(N_{CLB}, N_{i-Cluster}, N_{i-LUT}) + N_{i-Cluster} \times (\hat{K}_{MUX}(W) + \lceil \log_2 W \rceil \\ &\quad \times \hat{K}_{CB}) + \hat{K}_{MUX}(N_{clock}) + \lceil \log_2 N_{clock} \rceil \times \hat{K}_{CB}. \end{aligned} \tag{A24}$$

A.1.11. Configurable router

A configurable router has two different internal connections types, one for the crossing lines and another for the terminated lines. Fig. A8 depicts a router architecture wherein the crossing lines are connected with pass transistors (a), and the termination lines are connected with tri-state buffers (b).

The *PGB* is used for the terminating wires connections, and the Pass Transistors (Connection Point) are used for the crossing wire connections. The '*W*' parameter gives the total number of routing wires that are passing across the *routers* and the *logic clusters*. The '*s*' parameter defines the *Manhattan*[1] length of each wire. The given area model matches the general case whenever the router is located at an internal location. Whenever the router is located at the FPGA perimeter, the length of connecting wires is different from *s* in order to maintain the number of routing wires *W*. It can easily be shown that the number of terminating lines is *s* and therefore the number of passing wires is *W*-*s*. Each horizontal passing line is connected to its vertical pair that requires (*W*-*s*) connection points. Every terminating horizontal/vertical line pair requires 6 buffers; the total number of terminating pairs is *s*, so the total number of required buffers is 6s. Eq. (A25) summarizes the required area:

[1] Manhattan length is measured in Logic Cluster units. E.g. $s = 4$, whenever a connection line crosses over four Logic Clusters.

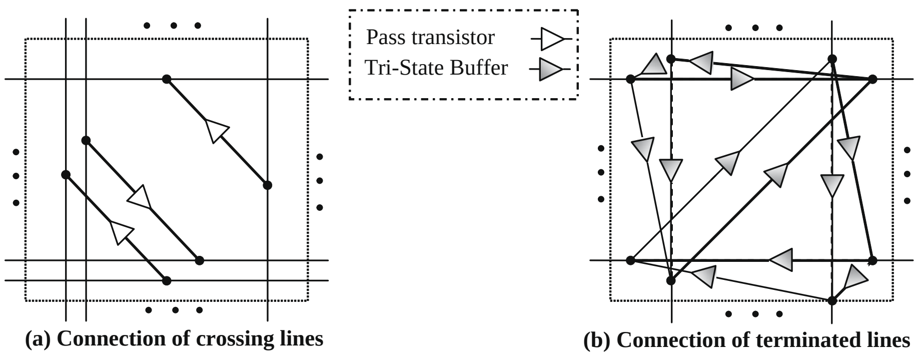

Pass transistor
Tri-State Buffer
(a) Connection of crossing lines
(b) Connection of terminated lines

Fig. A8. Configurable router architecture.

$$A_{Router}(W,s) = (W-s)\times A_{CP} + 6\times s\times A_{PGB}(s) = (W-s)\times 71.5 + 6\times s\times 91.5 = 71.5\times W + 457.3\times s. \quad (A25)$$

The static leakage power consumption of the router depends on the number of active configured buffers, due to different leakage model of active and inactive *PGB*. We define R_{active} be the ratio of the active configured buffers, as defined be $R_{\text{active}} = N_{\text{active}}/N_{\text{total}}$. The total power coefficient is thus:

$$\begin{aligned}\hat{K}_{Router}(W,s,R) &= (W-s)\times\hat{K}_{CP} + 6\times s\times(R\cdot\hat{K}^{active}_{PGB}(s) + (1-R)\cdot\hat{K}^{s\tan dby}_{PGB}(s))\\ &= (W-s)\times 29.2 + 6\times s\times\left(22\times s\times\frac{e^s-1}{e-1}\left(R+(1-R)\cdot\frac{1}{e}\right)+29.4\right)\\ &= 29.2\times(W-s) + 132\times s^2\times\frac{e^s-1}{e-1}\times\frac{1+R\cdot(e-1)}{e} + 176.4\times s. \quad (A26)\end{aligned}$$

A.1.12. Single context FPGA area and power models summary

This section summarizes the area and the static power coefficient estimation of a single-content FPGA. We present an FPGA wherein the logic blocks are arranged in a square. Therefore, every side has $\sqrt{N_{Cluster}}$ logic clusters, so the number of routers is given by $N_{Cluster} + 2\cdot\sqrt{N_{Cluster}}$. We want to simplify our model by reducing the number of parameters; therefore, we approximate some parameter values. For example, the average number of routing lines (W) could be calculated from the wire Manhattan length (s) and the number of inputs to logic block $N_{i\text{-}cluster}$. The approximation shown by [17] gives $W_{average} = \frac{N_{i-Cluster}\times s}{2}$. The number of inputs to the logic cluster, $N_{i\text{-}Cluster}$, is approximated by [16] as $N_{i-Cluster} = \frac{N_{i-LUT}}{2}\times(N_{CLB}+1)$. However, the occupied routing area varies with *logic cluster* size in a more complex manner. Whenever the cluster size grows, so does the length of the routing wires. As the routing wire length increases, additional repeating buffers must be inserted along each routing line. The area required by additional buffers is:

$$\begin{aligned}A_{buffers} &= 2\times\left(\left\lceil\sqrt{N_{CLB}}\right\rceil - 1\right)\times W_{Avg}\times\frac{\sqrt{N_{Cluster}}}{s}\times A_{2x},\\ A_{buffers} &= 2\times\left(\left\lceil\sqrt{N_{CLB}}\right\rceil - 1\right)\times N_{i-Cluster}\times\sqrt{N_{Cluster}}\times A_{2x}.\end{aligned} \quad (A27)$$

The final estimated area is:

$$\begin{aligned}A_{FPGA}(N_{i-LUT},N_{CLB},N_{Clsuter},N_{clock},s) &= N_{Cluster}\times A_{Cluster}\left(N_{CLB},\frac{N_{i-LUT}}{2}\times(N_{CLB}+1),N_{i-LUT},N_{clock},\frac{N_{i-LUT}}{4}\times(N_{CLB}+1)\times s\right)\\ &+\left[N_{Cluster}+2\cdot\sqrt{N_{Cluster}}\right]\times A_{Router}\left(\frac{N_{i-LUT}}{4}\times(N_{CLB}+1)\times s,s\right)\\ &+2\times\left(\left\lceil\sqrt{N_{CLB}}\right\rceil-1\right)\times N_{i-Cluster}\times\sqrt{N_{Cluster}}\times A_{2x}. \quad (A28)\end{aligned}$$

The power coefficient is derived in the same way:

$$\begin{aligned}\hat{K}_{FPGA}(N_{i-LUT},N_{CLB},N_{Clsuter},N_{clock},s,R) &= N_{Cluster}\times\hat{K}_{Cluster}\left(N_{CLB},\frac{N_{i-LUT}}{2}\times(N_{CLB}+1),N_{i-LUT},N_{clock},\frac{N_{i-LUT}}{4}\times(N_{CLB}+1)\times s\right)\\ &+\left[N_{Cluster}+2\cdot\sqrt{N_{Cluster}}\right]\times\hat{K}_{Router}\left(\frac{N_{i-LUT}}{4}\times(N_{CLB}+1)\times s,s,R\right)\\ &+2\times\left(\left\lceil\sqrt{N_{CLB}}\right\rceil-1\right)\times N_{i-Cluster}\times\sqrt{N_{Cluster}}\times\hat{K}_{2x}. \quad (A29)\end{aligned}$$

Table A4
Multi-context FPGA model parameters.

Symbol	Description
$N_{i\text{-}LUT}$	A number of inputs to LUT
N_{clock}	A number of clock trees
N_{CLB}	A number of CLB in a logic cluster
$N_{i\text{-}cluster}$	A number of inputs to a logic cluster
s	Manhattan length of the connection line
W	Number of wires in a routing segment
$N_{Cluster}$	Number of clusters in FPGA
N_c	Number of contexts
N_{config}	Number of configurable items in a device

Table A5
A hybrid MC-FPGA model parameters.

Symbol	Description
$N_{i\text{-}LUT}$	Number of inputs to LUT
N_{clock}	Number of clock trees
N_{CLB}	Number of CLB in logic cluster
s	Manhattan length of the connection line
$N_{Cluster}$	Number of clusters in configuration region
N_{Region}	Number of configuration regions
$N_{ConfPort}$	Number of configuration ports
$N_{DataPort}$	Number of data ports
$N_{DataSlice}$	Number of data slices
$N_{ConfSlice}$	Number of configuration slices
C_{Width}	Data width of the configuration bus

A.2. Local storage MC-FPGA model

In order to evaluate the area and the power coefficient of the multi-context devices, the previously described single context model is extended. The model has additional parameters that are summarized in Table A4.

We define two new parameters N_c and N_{config}. The area and power coefficients are derived by using the $N_{config} = N_{cluster} \times N_c$ relation.

The complete model derivation can be found in [23] Section 6.3.

A.3. Hybrid MC-FPGA model

The previously presented area and power estimation models for the dual-context architecture is extended. In order to describe the new area and power models, additional parameters are defined in Table A5.

The complete model derivation can be found in [23], Section 6.5.

Appendix References. (Appendix Only)

[A1]. N.H.E. Weste, D. Haris, “CMOS VLSI design – A circuit and system Perspective, 3rd Edition”, Pearson Addison-Wesley, 2005.
[A2]. M. C. Johnson, D. Somasekhar, K. Roy, “Models and Algorithms for Bounds on Leakage in CMOS Circuits”, IEEE Tran. on computer-aided design of integrated circuits and systems, p. 714, Vol. 18, No. 6, June 1999.
[A3]. “Integrated Circuit – Introduction to VLSI (046237)”, Graduated course slides, Technion – Israel institute of Technology, Fall 2007.

Appendix B. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.compeleceng.2008.11.024.

References

[1] Intel® Ltd., Introducing the 45 nm next-generation Intel® Core™ micro-architecture, White Paper, <http://www.intel.com/technology/architecture-silicon/intel64>, 2007.
[2] Venkateshan A, Singh R, Poole KF, Harriss J, Senter H, Teague R, et al. High-k gate dielectrics with ultra-low leakage current for sub-45 nm CMOS. Electron Lett 2007;43(21):1130–2.
[3] Eitan B, Pavan P, Bloom I, Aloni E, Frommer A, Finzi D. Can NROM, a 2-bit, trapping storage NVM cell, give a real, challenge to floating gate cells? Saifun Semiconductors Ltd.; 1999.
[4] Trimberger S, Carberry D, Johnson A, Wong J. A time-multiplexed FPGA. IEEE Sym FPGA-Based Custom Comput Mach 1997(April).
[5] Tau E, Chen D, Eslick I, Brown J, DeHon A. A first generation DPGA implementation, third canadian workshop of field-programmable devices, June 1995.
[6] © 2002–2005 Xilinx, Inc., Virtex-II Pro and Virtex-II Pro X Platform FPGAs: functional description, <www.xilinx.com>, March 2007.
[7] DeHon A. DPGA utilization and application. In: Proceedings of the fourth international ACM symposium on field-programmable gate arrays; 1996.
[8] Scalera SM, Vazquez JR. The design and implementation of a context switching FPGA. In: Proceedings of the IEEE Symposium on FPGAs for custom computing machines; April 1998.
[9] Compton K. Programming architectures for run-time reconfigurable systems. M.Sc. Thesis, Evanston, IL, USA: Northwestern University; December 1999.
[10] Enzler R, Plessl C, Platzner M. Virtualizing hardware with multi-context reconfigurable arrays. In: Proceedings of the 13th international conference on field programmable logic and applications; September 2003.
[11] Semiconductor Industry Assoc., International technology roadmap for semiconductors, 2002 Update, <http://www.electrochem.org/dl/ma/201/pdfs/0552.pdf>.
[12] Kim NS, Austin T, Blaauw D, Mudge T, Flaunter K, Hu JS, et al. Leakage current: Moore's law meets static power, computer, vol. 36. IEEE Press; 2003. p. 68–75.
[13] Butts JA, Sohi GS. A static power model for architects. In: Proceedings of the 33rd annual international symposium on micro-architecture, MICRO'33; December 2000.
[14] Ko P, Huang J, Liu Z, Hu C. BSIM3 for analog and digital circuit simulation. In: Proceedings of the IEEE symposium on VLSI technology CAD; January 1993. p. 400–29.
[15] Betz V, Rose J. Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size. In: Proceedings of the IEEE custom integrated circuits conference; May 1997.
[16] Ahmed E, Rose J. The effect of LUT and cluster size on deep submicron FPGA performance and density. IEEE Trans VLSI syst 2004(March).
[17] Kuon I. Automated FPGA design, verification and layout. Master Thesys, University of Toronto; 2004.
[18] Weste NHE, Haris D. CMOS VLSI design – a circuit and system perspective. 3rd ed. Pearson Addison-Wesley; 2005.
[19] Kuon I, Egier A, Rose J. Design, layout and verification of an FPGA using automated tools. In: Proceedings of the international symposium on Field-programmable gate arrays (FPGA'05); February 2005.
[20] Chang YW, Lu TC, Pan S, Lu CY. Modeling for the 2nd-Bit effect of a nitride-based trapping storage flash E2PROM cell under two-bit operation. IEEE Electron Device Lett 2004;25(2):95–7.
[21] Jeong H, Song KW, Park IH, Kim TH, Lee YS, Kim SG, et al. A new capacitorless 1T DRAM cell: surrounding gate MOSFET with vertical channel (SGVC cell). IEEE Trans Nanotechnol 2007;6(3):352–7.
[22] IEEE Std. 1149.1-1990 IEEE standard test access port and boundary-scan architecture. <http://standards.ieee.org/reading/ieee/std_public/description/testtech/1149.1-1990_desc.html>.
[23] Fiksman E. Dynamic reconfiguration architectures for multi-context FPGAs. M.Sc. Thesis, electrical engineering department. Haifa, Israel: Technion – Israel Institute of Technology; March 2008.
[24] XtremeData, Inc. XD2000i™ FPGA IN-SOCKET ACCELERATOR for INTEL FSB. Product description, <http://www.xtremedatainc.com/index.php?option=com_content&view=article&id=54&Itemid=58>.

Yitzhak (Tsahi) Birk received the B.Sc. (cum laude) and M.Sc. degrees from the Technion in 1975 and 1982, respectively, and the Ph.D. degree from Stanford University in 1987, all in electrical engineering. He is an Associate Professor in the Electrical Engineering Department at the Technion, and heads its Parallel Systems Laboratory (PSL). From 1976 to 1981, he was project engineer in the Israel Defense Forces. From 1986 to 1991, he was a Research Staff Member at IBM's Almaden Research Center, where he worked on parallel architectures, computer subsystems and passive fiber-optic interconnection networks. From 1993 to 1997, he also served as a consultant to HP Labs in the areas of storage systems and video servers, and was later involved with several companies. His research interests include computer systems and subsystems, as well as communication networks. He is particularly interested in parallel and distributed architectures for information systems, including communication-intensive storage systems (e.g., multimedia servers) and satellite-based systems, with special attention to the true application requirements in each case. The judicious exploitation of redundancy for performance enhancement in these contexts has been the subject of much of his recent work. He is also engaged in research into various facets of processor architecture, attempting "cross fertilization" between his various areas of research.

Evgeny Fiksman received the B.Sc. degree in computer engineering from the Technion – Israel Institute of Technology, in 1999. He is currently pursuing the M.Sc. degree in electrical engineering at the Technion. Since March 2000 he has been an undergraduate project supervisor at the Parallel Systems and the High-Speed Digital Systems laboratories. From 1999 to 2006, he was project engineer in the Israeli Navy. Since April 2006 he has worked at Intel Development Center in Haifa, Israel. His research interests include areas of FPGA architectures, SoPC (System on a Programmable Chip) architectures and reconfigurable computing.