Challenges of Electronic CAD in the Nano Scale Era

Christian Hochberger
Chair for Embedded Systems
TU Dresden
christian.hochberger@tu-dresden.de

Andreas Koch
Embedded Systems and Applications Group
TU Darmstadt
koch@esa.cs.tu-darmstadt.de

Abstract: Future nano scale devices will expose different characteristics than todays silicon devices. While the exponential growth of non recurring expenses (NRE, mostly due to mask sets) can be anticipated even for new technologies, problems such as the dramatically increased defect density require new approaches to build functional devices at reasonable prices. Improved CAD algorithms can help to solve these problems, or in some cases, they can be seen as enabling technology to broaden the use of paradigms such as reconfigurable computing. In this work we discuss in which stages of design, manufacturing, and deployment new CAD algorithms are required.

1 How to make Productive use of Billions of Logic Gates

Following the road of Moore's law, the number of transistors on a chip doubles every 24 months. After being valid for more than 40 years, the end of Moore's law has been forecast many times now. Yet, technological advances have keep the progress intact.

While the technological forecast for the next 5 to 10 years still concentrates on traditional CMOS logic realized on silicon, it seems likely that other technologies will take over in this time frame. A good candidate is CMOL[SFG$^+$03][DL05], which uses carbon nano tubes together with silicon-implemented CMOS circuits. With this technology, logic elements can be built at a much higher density than possible with lithographic processes using etching to build up physical structures.

In the last year, graphene films have been found as another candidate technology for future digital devices[DSMJ09]. In contrast to CMOL, no actual logic devices have been built yet, leaving the design and cost implications unclear. However, it can be speculated that the extremely small structures used here will also be susceptible to higher defect rates.

1.1 Consequences

Further shrinking of the feature size of traditionally manufactured chips causes two major problems.

1. Mask costs increase exponentially. This makes it prohibitively expensive to produce

small quantities of chips for one particular design.

2. Defect density increases dramatically. This reduces the yield of the chip production.

Unfortunately, these problems will not be solved by alternative technologies. In fact, the defect density will increase even more dramatically in CMOL structures. Here, a large number of the nano tubes will not be functional. Thus, it is practically impossible to manufacture defect-free devices of reasonable size.

Although well known fault tolerant design techniques exist, it is extremely tedious to harden designs against defects. Also, these techniques lead to a considerable increase in area consumption and might also cause a performance degradation (think of the notorious triple modular redundancy). Thus, these techniques might not be practical for large designs or for high defect rates, which can be expected from technologies like CMOL.

1.2 Current philosophy to deal with these Problems

In short, we can conclude at this point that the two major problems for chip design and manufacturing are the dramatically increasing defect rates and the exponentially growing mask costs. We will now give a brief review of the current strategies to solve these problems.

1.2.1 Defect minimization

Defects are currently mostly considered at the chip production stage. Manufacturers try to minimize the defect density with more precise lithography, improved clean room conditions and thorough process monitoring. Beyond the production stage, few attempts have been made to tolerate higher defect densities. Typically, designers include spare elements (logic, memory, processor cores, ...) that can take over from defective parts. Devices are post processed and "trimmed" in the factory to appear fully functional. We will discuss this approach in much more detail in section 2.1.1.

These approaches will not work for defect rates that are several orders of magnitude higher than today.

1.2.2 (Re)configuration

The exponentially increasing cost of masks lead to larger minimum quantities for an effective chip production. This can be achieved by using one chip for more than one purpose. A well known method to enable this multi-purpose use is (re)configuration. Configuration can be seen as a structural programming. It means to use one and the same structural element for different purposes in different configurations. Reconfiguration refers to the ability to change a configuration of a device during runtime[1].

[1]Other definitions for configuration and reconfiguration might exist. We use this relatively relaxed definition as it covers the widest range of applications.

Reconfiguration can be applied to structural elements at different granularities. One can use bit level elements resulting in Field Programmable Gate Array (FPGA)-like devices. Here, the reconfigurable elements typically implement logical operations on the boolean inputs. Alternatively, one can use word (or sub-word) level elements resulting in so called coarse-grain reconfigurable arrays (CGRAs). In this case, the reconfigurable elements typically implement arithmetic operations that might even include multiplication or division. FPGAs are already well established in digital circuit design and have captured many application areas (especially those with smaller volumes). In contrast, CGRAs are a topic of current research, but are still awaiting some commercial success.

1.3 Paper Outline

In the following section, we will discuss general approaches that can be used to solve the two prevalent problems. In the third section we will further discuss which contribution enhanced CAD algorithms can make to these approaches. Finally, we will give a short conclusion.

2 General Approaches

Given the characteristics of nano-scale devices described in the previous Section, we will now discuss some of the issues that must be addressed.

2.1 Handling of defective devices

Regardless of the actual base technology (e.g., nano-scale CMOS or "true" nano-technologies such as carbon nano tubes, nano wires, or graphene films), the fabricated chips will have a much higher defect density than the previous generation of deep sub-micron devices. In this section, we will examine how this can be handled in practical usage.

2.1.1 Fab-side hiding of general defects

The first approach tries to preserve the illusion of defect-free devices (at least in the view of the designer) by providing sufficient redundancy on the die (below the architectural level). Fab-time tests discover the actual defects present on each chip. Then, techniques such as laser or e-beam cutting (breaking connections) or programming anti-fuses (establishing connections) are employed to activate redundant resources to specific defects.

Such techniques are already in practical industrial use to increase the yields of 100% visible defect-free devices for DRAMs [FHM$^+$05] and FPGAs [YL05] [Alt] [CCCV06].

However, the growing defect densities of nano scale devices would require excessive de-

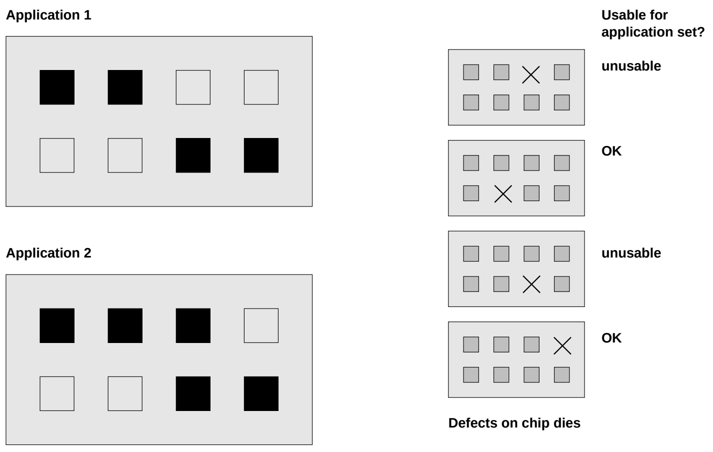

Figure 1: Fab-side defect hiding for limited application set

grees of redundancies to keep up this illusion, raising chip costs to uneconomical levels. Note that growing redundant areas would themselves be subject to an increasing number of defects.

Thus, in the nano scale era, chip defects *will* be visible beyond the fab line and will need to be dealt with in a more specialized manner than sheer redundancy.

2.1.2 Application-specific defect handling

A. For a limited set of applications The scope of 100% defect-free functionality can be narrowed from all (user-visible) resources on the chip to just those required for individual (or small sets of) *applications* (see Figure 1).

For current deep sub-micron CMOS devices, this is already being done at an industrial level for up to two applications [Xil08]. In these programs, the vendors are provided with a complete mapping of the application(s) to the target device (generally in the form of completely pre-placed and pre-routed bit streams). Thus, the specific resources required for the mapped designs are precisely known. For this design, the requirement of totally defect-free chips (as would be required for general use) can now be relaxed to just being defect free for the *specific application mapping*. Note that this strategy does incur some NRE costs for setting up application-specific chip testing ($<$ USD 100K). Also, this is only feasible for a small number of applications. More applications would most likely stress a larger number of different resources on the device, thus increasing the chance of defects becoming visible (and lowering the effective yield).

With specialized CAD tools, this effect could be reduced, however: One could design the implementation tool flow (mapping, placement and routing) in such a manner that multiple applications are processed to employ the minimum number of *different* resources on the

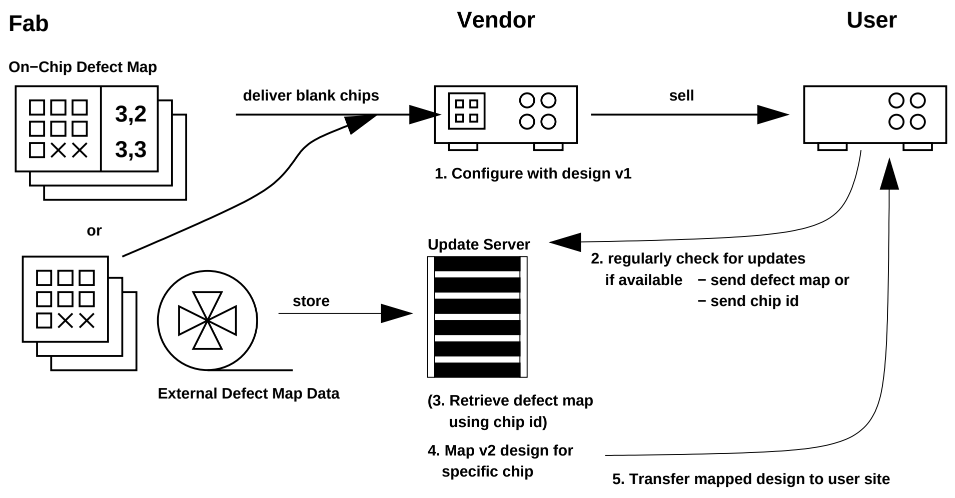

Figure 2: Vendor-side customization of user-site hardware

chip. This would both increase the chance of application set-specific defect free devices as well as reduce the chip test overhead (which is proportional to the number of resources tested). Research on such algorithms has been performed for speeding-up dynamic partial reconfiguration (only resources that are different between two successive configurations have to be reconfigured) [JK08][HMW04], they could be adapted for defect-handling purposes.

B. For general applications, customized vendor-side If a larger set of different applications has to be mapped to potentially defective devices, at some point it becomes necessary to compute *device-specific configurations* (instead of selecting configuration-specific devices, as discussed above).

In contrast to the dynamic approach discussed in Section 2.2.2, this can also be statically performed vendor-side and does not necessarily require chip fab assistance. In this scenario, shown in Figure 2, a solution vendor would buy unconfigured devices from a fab that have an agreed-upon minimum logic capacity, but also an essentially random distribution of defects. For each device, the vendor would either be provided with their location in the form of a *defect map* by the fab, or test the devices itself. The defect map could be provided within each device (e.g., in the form of a robust high-density PROM, written at the fab), or externally to it (e.g., per-lot data files).

Assuming that the frequency of updates to the applications remains reasonably low (on the order of weeks to months, an interval typical for in-the-field software updates), the following approach could be viable: When a software update necessitates an update of the device configurations, the user in the field connects to the vendor-side update server. Either a device ID (referencing the defect database at the vendor) or the complete on-device defect map itself is transferred to the vendor. At the vendor, fast CAD tools running on a highly parallel compute farm of fast servers now map, place and route the new version of each application to the device, omitting its specific defects. The device-specific bit streams are then transmitted to the user site for installation. This entire process can be encapsulated in automatic updater programs and be performed transparently to the user.

Note that the need for fast CAD tools in this scenario is different from that outlined in Section 2.2.2. For the vendor-side remapping described here, the term “ fast” describes a time-interval acceptable to the user-site updater (which can range from a few-minutes to even days for automatic updaters running in the background). Also, the compute environment for customizing the applications to the device (large compute farm) is significantly different from that of 2.2.2 (embedded computing).

C. For general applications, customized at the user-site As a mid-point between devices appearing to be 100% defect-free (A., redundant resources activated fab-side) and the static customization of bit-streams for individual chips (B., vendor-side) it is possible to expose a limited set of redundancy (e.g., just 20% of additional tracks) to a user-side flow. Contrast this with the prior approaches, where we either had enough redundancy to appear totally defect-free (A.), or no explicitly designated redundancy at all (B., we just ran the complete implementation flow to work around defects using the generic, but functional on-chip resources).

In the approach proposed in [RD09], a vendor-provided bit stream for an application contains not just one mapping of a configuration, but also has alternate routes for each point-to-point connection that use one of the redundant wires (called reserved wires in [RD09]). Defective wires are determined by user-side testing. In this fashion, a single bit stream can be used to successfully configure all devices that have a defect density in the routing fabric less than the provided redundancy.

This approach does not require vendor intervention into the device-specific customization process. Instead, it is performed at bit stream load-time on the user-site. However, this flexibility comes at the price of both increased bit stream sizes and load times (by over 40x and 160x, respectively [RD09], even for modest applications).

2.2 Increasing fabrication volume

As discussed in the Introduction, the exponentially rising NRE costs when fabricating a new chip design (e.g., for mask preparation) allow the use of cutting-edge fab processes only for extremely large production volumes. Reconfigurable devices are very suitable for this scenario, since they remain general-use parts even after fabrication. While reconfigurable devices have been employed successfully (in terms of performance, energy efficiency, etc.) in a wide variety of fields [LR05], they are still not as ubiquitous as would be desirable to exploit the economies-of-scale on extremely expensive fab processes.

The most often given reason for not using reconfigurable devices, even when doing so would result in improved performance or energy efficiency, is the difficulty of actually porting an application away from software to run on the device instead. Today, this process involves languages unfamiliar to a software developer (e.g., HDLs such as Verilog and VHDL) as well as a familiarity with digital logic design and computer architecture.

Thus, in order to significantly increase chip production volumes of reconfigurable devices, it becomes essential to make the technology actually accessible to a wider user-base of non-hardware design experts. The research in and development of appropriate CAD tools and models of computation thus becomes crucial for the entire economic viability of nano-scale chip fabrication (and not just the ease of implementing a single application on a reconfigurable device, as today!).

2.2.1 Static compilation to reconfigurable devices

Numerous past and present research efforts aim at raising the abstraction level of "programming" a reconfigurable device from hardware-like concepts (gates, registers, etc.) to software-like levels [BVCG04] [KK05] [PBD^{+}08]. Often, these flows accept as input a conventional software programming language such as C (though usually just a subset), and automatically translate the semantics of the C program into efficient hardware structures suitable for the reconfigurable device.

The first of these flows have been used in production settings [Men09] and research continues to advance both in the degree of language features supported (e.g., arrays, pointers, irregular control flow) as well as the efficiency with which they can be translated into hardware structures (e.g., automatic recognition of regular data streams and their mapping to DMA engines instead of caches).

Many of these flows have in common that their results achieve very high peak performance only for a limited number of language constructs (e.g., an inner loop with just affine index calculations, containing neither control flow and nor pointer operations in its body). While a large number of important programs (e.g., from digital signal or media processing) does indeed fall in this category, and can thus be successfully accelerated, many other general-purpose applications (word processing, compilers, etc.) contain more convoluted data and control structures and are only translated in a very limited fashion (if at all) for hardware execution on a reconfigurable device.

If our aim is to make reconfigurable devices attractive compute elements for a much wider range of applications, thus leading to the massively increased production volume crucial for the economical use of nano-scale fabrication processes, we need to look *beyond* static translation techniques.

2.2.2 Dynamic compilation to reconfigurable devices

Dynamic translation examines an *executing* software program and generates hardware structures for frequently executing instruction sequences *without* regard for the high-level language structures they originated from. This has the disadvantage that some high-level language structures for which very efficient hardware realizations can be created (e.g., the DSP-style loop described for static compilation) might not be recognized from the instruction stream. But now *all* suitable instruction sequences, regardless of their origin, can be translated to an appropriate hardware accelerator in a just-in-time manner.

While it is unlikely to reach the peak-performance achievable by powerful static compilers employing, e.g., whole-program analysis and optimization, dynamic hardware compilation has a number of advantages. First, since the process can operate from both machine code sequences (e.g., generated from C) as well as byte codes (such as generated by Java or the .NET languages). Thus software developers may continue to use the programming languages they are most proficient in, without worrying about the hardware aspects of the implementation.

Second, from the hardware perspective, the instruction or byte code sequences captured during execution act as a portable intermediate format. The *same* binary can be executed on a later model reconfigurable device (possibly larger and faster), and profit from the increased compute capability without user or developer intervention. This is similar to the approach proposed by OpenCL [Khr09], which translates the actual *source code* of compute kernels at run-time into the most efficient form for the *current* execution platform (one or more CPUs, zero or or more GPUs).

Third, since no hardware/software partitioning takes place in advance, the characteristics of the *current* execution (e.g., input data characteristics, user actions, etc.) are used to guide the hardware translation, not a single profile captured during a static compilation flow. Instead of the static cherry-picking of just promising compute kernels, the dynamic translator works on a much finer granularity and potentially moves a larger fraction of the executing program to hardware acceleration. This makes the approach attractive for the handling of general-purpose programs that often lack the structures focused on by the static compilers.

Fourth, the dynamic generation of hardware structures (which will be discussed in greater detail in Section 3.2) is perfectly suited to mapping around the device-specific defects inherent in the use of nano-scale technologies. Thus, the difficulties and possibly complex solutions discussed in Section 2.1 (e.g., large customization compute farms at vendors, etc.) can be avoided.

The feasibility of this approach has been shown in the WARP processor [VSL08, LV09], which extracts hardware structures from ARM instruction sequences at runtime. WARP demonstrated not only the successful extraction, but showed that the entire hardware implementation flow (mapping, placement, routing) could be performed on-the-fly with sufficient efficiency. More ambitious attempts [GH05] aim to go beyond WARP's limitations (e.g., mapping sequences to just combinational operators on a very simple reconfigurable device architecture).

3 Open Problems

3.1 Effects of granularity

Choosing the right granularity for the reconfigurable fabric is a question that might be answered differently in light of the problems that the nano scale era imposes on the designer.

3.1.1 New fine grain logic structures

Today, fine grain logic is typically implemented as lookup tables (LUTs). A small number of LUTs is combined into one larger element called configurable logic block or logic array block. This hierarchy allows reduced external connectivity while preserving high connectivity internally. Defects that occur in LUTs can only be *healed* by replacing them with spare LUTs. Also, defective routing resources can only be healed by replacing them with spare resources. Unfortunately, this does not only require additional area for the spare route, but also for the additional connection points. Thus, it seems to be very inefficient to implement a considerable amount of redundancy using current FPGA architectures.

Alternatively, researchers have already looked at other building blocks. Programmable logic arrays (PLAs) have been investigated and seem to solve some problems[DN05]. A nanoPLA based on carbon nano tubes is depicted in figure 3. The two rows each form a NOR plane. Both NOR planes are cyclically connected, such that signals can be propagated several times through a single PLA. Horizontal wires can be used as input and output. Redundancy can be provided in this case at a smaller granularity. PLAs can include spare rows and columns. Also, re-routing becomes easier, since inputs as well as outputs of PLAs are almost fully exchangeable. Remapping of an existing circuit under defect conditions becomes almost trivial in this case. Yet, this type of architecture introduces new synthesis problems, since the mapping of general circuits onto a network of PLAs is not well understood currently. Thus, technology mapping onto PLAs is one of the open CAD problems that need to be solved.

As the field of research in alternative fine grain architectures is not very popular today, it might well happen that researchers come up with totally different building blocks. But even then it is highly improbable that existing mapping algorithms can be used.

3.1.2 On-chip test capabilities

Some approaches to the handling of defect devices require that the chip itself can evaluate the logic and routing resources that are fully operational. Current architectures rely on external testing to evaluate the defect/operational resources. Providing efficient internal measures to probe for defect/operational resources is an open problem that is closely related to the architecture of the building blocks. Some research has been done with respect to LUT based elements, but in our opinion this is not sufficient, especially, with much higher defect rates.

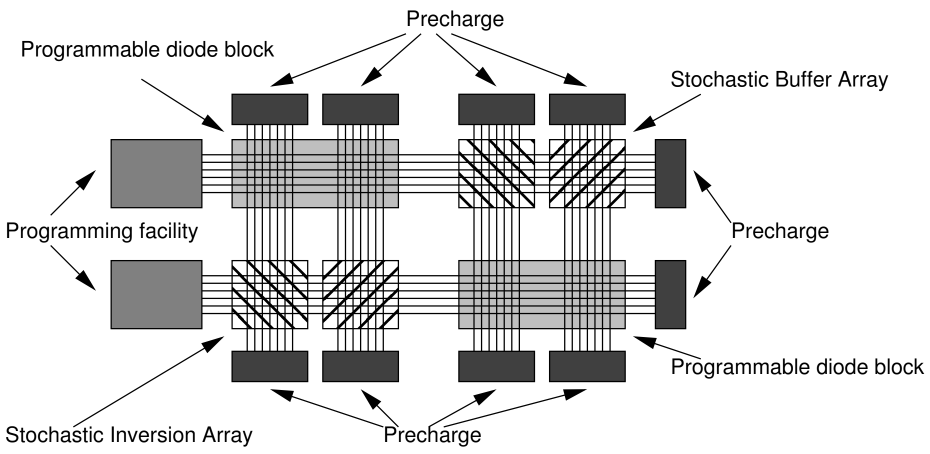

Figure 3: A nanoPLA composed of two rows of NOR planes

3.1.3 Defect tolerant coarse grain architectures

By definition, CGRAs use more complex elements as building blocks. Thus, these blocks use much larger numbers of transistors (or whatever element the base technology provides to implement logical functions). Consequently, the probability of a defect in such an element is much higher than in fine grain logic. To reach a sufficient amount of yield, it will be necessary to implement defect-tolerant coarse grain elements. To the best of our knowledge, no research has been carried out in this area up to now (while this is not directly a topic of CAD, it deserves to be mentioned here).

As the number of nodes that have to be handled by mapping, placement and routing is at least two orders of magnitude smaller than with fine grain logic, we can afford to tolerate a reasonable large number of defective processing elements in the CGRA fabric. The much smaller complexity allows us to map and place applications *around* defective nodes. This relaxes the requirements on the defect tolerance of processing nodes. Nevertheless, we still need to apply architectural improvements to reach a sufficiently high number of defect-free processing nodes.

3.2 Using dynamic SW/HW migration to broaden applicability

As mentioned in section 2.2.2, dynamically selecting code sequences at runtime as candidates for a hardware mapping has several benefits. In this case, the control and data flow has to be extracted from the execution binary of the underlying software platform.

Most of the current research in this area is focused on producing *better* results, often at the expense of longer runtime or larger memory requirements. In the case of dynamic compilation, however, we are pursuing a different goal. We value shorter run time, trading it for a lower result quality. In general, this goal can be captured by the following phrase: Trade 10% result quality for a factor of ten reduction in the runtime. Empirical observations made with the state of the art place & route tool VPR[BRM99] show that, at least for FPGA placement, this aim is achievable.

3.2.1 Mapping of applications

The control and data flow that has been extracted from application code may contain arbitrary operations. Also, it may contain alternative execution branches whose direction depends on runtime data (`if-else` statements). These structures have to be mapped on the CGRA in such a way that complex operations, which might not be available natively, are decomposed into a set of simpler operations. Furthermore, alternative execution branches are not desireable, especially in the body of loops. Two concepts can be used to avoid real control flow in this case. 1.) Both branches might be executed and the appropriate result is selected at the end of both computations. Additional measures have to be taken to avoid unnecessary waiting for the complementary execution path if it is not required. 2.) Operations can be predicated, meaning that they are attributed with conditions. If the conditions are met, the operation is carried out. Typically, one of the operands is passed through the operation unchanged if the condition is not met. In general, both variants require modifications of the control and data flow.

In case the CGRA is not homogeneous, not all operations can be assigned to all nodes. Thus, mapping is interdependent with placement and routing.

A further complication that needs to be addressed by the mapping algorithm is the pipelining of loop bodies. Simply mapping the body of a loop onto a CGRA typically leads to relatively poor performance and utilization. Execution of loop bodies can be overlapped and, in a second step, the loop can be partially unrolled to extend the scope of mapping and improve utilization.

None of today's most advanced mapping approaches take into account all options discussed here[**?**]. For example the DRESC framework[MVV$^+$03] neither considers the decomposition of operations, nor is it able to unroll loops. Also, most current mapping approaches require enormous amounts of computing time, making them infeasible in a dynamic compilation environment.

3.2.2 Placement of processing elements

Placement and routing are NP complete problems. Thus, most approaches today use some heuristics to avoid this bottleneck. A very popular heuristic here is simulated annealing. It involves the exchange of nodes in the array and the computation of a cost function. Since these steps are repeated many times, it is a natural idea to speed up this process by executing it on dedicated hardware. For FPGAs, this approach has already been initially investigated with good success[WD03]. We are not aware that a similar approach has been used on CGRAs, though. Alternatively, other heuristics could also be employed for this problem. Genetic algorithms seem to work well for smaller problems and could also profit from hardware support.

Both approaches cannot be parallelized easily. Thus, even with hardware support, these placement algorithms might take too long in a dynamic compilation environment. A totally different approach could try to optimize the placement locally in a cellular automaton-like structure. Each cell of the cellular automaton would handle a small number of processing elements and exchanges processing nodes with their direct neighbours. The supporting hardware could be implemented directly in the CGRA fabric would thus scale much better with the size of the CGRA. Additionally, this approach could be enhanced with some kind of supervisory process that could observe global parameters and would adjust local control parameters in each cell in order to achieve a global balance of cells.

3.2.3 Routing of data

The routing problem for reconfigurable devices has long been studied in context of traditional FPGAs. The core problem consists of successfully routing all required connections using the limited resources on the device. The Pathfinder algorithm [ME95] has been extremely successful for this application, it is used as the heart of many routing tools (both academic and industrial). Pathfinder relies on incrementally increasing the cost of "popular" resources (preferred by many nets) to drive nets that do have acceptable alternatives away from the congested resource until no resource on the device is overextended.

For the nano-scale devices discussed here, two additional aspects need to be examined. First, the algorithm can be extended to work around defective resources. This is used in [RD09], where the router generates alternatives for an assumed maximum number of defects in each routing segment. These alternatives are then available at bitstream load time to route around the defects on the specific device (see section 2.1.2.C).

Second, the dynamic generation of hardware (as suggested in section 2.2.2 as a means to increase fabrication volume) becomes only feasible with fast-running algorithms. With the memory and compute requirements of Pathfinder (due to storing and repeatedly searching the graph consisting of every routing resource present on the device), it appears that it is unsuitable under these conditions. However, this problem can be tackled in a number of ways: Since we are mainly interested in CGRAs, we can argue that the switching of entire N-bit busses requires storing/processing only of $1/N$ of nodes in the routing graph (compared to that of an FPGA that routes each bit individually).

Then, as suggested in section 3.2.2, we can dedicate some of our hardware area to accelerate the routing process. Accelerators for routing have been suggested a number of times [Ios86, RR87, WKS87]. The most promising approach for nano-scale devices, however, appears to be that used in [DHW06], where the routing fabric of the device itself has been extended to perform the required traversals of the routing graph in parallel. Thus, with increasing device size, the degree of parallelism scales correspondingly. Even for fine-grained FPGAs, this approach has resulted in speed-ups of five to six orders of magnitude over fast router implementations in software.

Note that the same idea, using the fabric to represent itself in CAD algorithms, could also be employed for generating the defect map that the defect-avoidance steps of the implementation steps require.

3.2.4 Combined approaches

As discussed in the previous sections, mapping, placement and routing may be highly interdependent, especially if the processing elements and their connections are not fully homogeneous. Thus, it is a natural idea to combine all three of these steps into one generalized optimization problem. The evaluation of legal configurations is more complex in this case than when the three problems are solved separately. Thus, the scope of such combined approaches is typically reduced to loop kernels of moderate size. Possible approaches to solve the underlying optimization problems are either exact solvers such as integer linear programming, or heuristic solvers such as simulated annealing (used in the DRESC framework[MVV$^+$03]). Both approaches have very high runtime requirements, thus they seem unsuitable for dynamic compilation. Here, new heuristic solutions are required, providing a better balance between runtime and result quality.

4 Conclusion

In this work we have shown that the two prevalent problems of chip production in the nano scale era can only be solved by new tools and methods in design and application. We have shown that the expected defect density requires new strategies and architectures, especially for configurable logic. These strategies and architectures are highly dependent on appropriate CAD algorithms. We have also shown that, besides the traditional use of static reconfiguration, a more dynamic approach promises a wider applicability. This dynamic use of reconfiguration also requires suitable CAD algorithms. It is our belief that research for these new CAD algorithms must start today in order for them to be at hand when real nano scale devices become available in commercial quantities.

References

[Alt] Altera Corp. APEX Redundancy: The Altera Advantage.

[BRM99] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs.* Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[BVCG04] Mihai Budiu, Girish Venkataramani, Tiberiu Chelcea, and Seth Copen Goldstein. Spatial computation. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 14–26, New York, NY, USA, 2004. ACM.

[CCCV06] Nicola Campregher, Peter Y. K. Cheung, George A. Constantinides, and Milan Vasilko. Yield enhancements of design-specific FPGAs. In *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 93–100, New York, NY, USA, 2006. ACM.

[DHW06] André DeHon, Randy Huang, and John Wawrzynek. Stochastic spatial routing for reconfigurable networks. *Microprocessors and Microsystems*, 30(6):301–318, 2006.

[DL05] André DeHon and Konstantin Likharev. Hybrid CMOS/nanoelectronic digital circuits: devices, architectures, and design automation. In *ICCAD*, pages 375–382. IEEE Computer Society, 2005.

[DN05] Andre DeHon and Helia Naeimi. Seven strategies for tolerating highly defective fabrication. *Design & Test of Computers, IEEE*, 22(4):306–315, July-Aug. 2005.

[DSMJ09] Sujit S. Datta, Douglas R. Strachan, E. J. Mele, and A. T. Charlie Johnson. Surface Potentials and Layer Charge Distributions in Few-Layer Graphene Films. *Nano Letters*, 9(1):7–11, 2009.

[FHM$^+$05] Kiyohiro FURUTANI, Takeshi HAMAMOTO, Takeo MIKI, Masaya NAKANO, Takashi KONO, Shigeru KIKUDA, Yasuhiro KONISHI, and Tsutomu YOSHIHARA. Highly Flexible Row and Column Redundancy and Cycle Time Adaptive Read Data Path for Double Data Rate Synchronous Memories. *IEICE Trans Electron*, E88-C(2):255–263, 2005.

[GH05] Stephan Gatzka and Christian Hochberger. The AMIDAR Class of Reconfigurable Processors. *The Journal of Supercomputing*, 32(2):163–181, 2005.

[HMW04] Lei He, T. Mitra, and Weng-Fai Wong. Configuration bitstream compression for dynamically reconfigurable FPGAs. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 766–773, Washington, DC, USA, 2004. IEEE Computer Society.

[Ios86] Alexander Iosupovici. A Class of Array Architectures for Hardware Grid Routers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 5(2):245–255, 1986.

[JK08] Sungjoon JUNG and Tag Gon KIM. Configuration Sharing to Reduce Reconfiguration Overhead Using Static Partial Reconfiguration. *IEICE Trans Inf Syst*, E91-D(11):2675–2684, 2008.

[Khr09] Khronos Group. OpenCL Specification. 2009.

[KK05] Nico Kasprzyk and Andreas Koch. High-Level-Language Compilation for Reconfigurable Computers. In *Proc. Intl. Conf. on Reconfigurable Communication-centric SoCs (ReCoSoC)*, 2005.

[LR05] Patrick Lysaght and Wolfgang Rosenstiel. *New Algorithms, Architectures and Applications for Reconfigurable Computing*. Springer, July 2005.

[LV09] Roman Lysecky and Frank Vahid. Design and implementation of a MicroBlaze-based warp processor. *ACM Trans. Embed. Comput. Syst.*, 8(3):1–22, 2009.

[ME95] Larry McMurchie and Carl Ebeling. PathFinder: A Negotiation-based Performance-driven Router for FPGAs. In *FPGA*, pages 111–117, 1995.

[Men09] Mentor Graphics Corp. High Level Synthesis – Catapult-C. 2009.

[MVV$^+$03] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins. Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling. *Computers and Digital Techniques, IEE Proceedings -*, 150(5):255–61–, Sept. 2003.

[PBD$^+$08] Andrew Putnam, Dave Bennett, Eric Dellinger, Jeff Mason, Prasanna Sundararajan, and Susan J. Eggers. CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures. In *FPL*, pages 173–178. IEEE, 2008.

[RD09] Raphael Rubin and André DeHon. Choose-your-own-adventure routing: lightweight load-time defect avoidance. In *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 23–32, New York, NY, USA, 2009. ACM.

[RR87] Thomas Ryan and Edwin Rogers. An Isma Lee Router Accelerator. *IEEE Des. Test*, 4(5):38–45, 1987.

[SFG$^+$03] Mircea R Stan, Paul D Franzon, Seth Copen Goldstein, John C Lach, and Matthew M Ziegler. Molecular Electronics: From Devices and Interconnect to Circuits and Architecture. *Proceedings of the IEEE*, 91(11), November 2003.

[VS07] Stamatis Vassiliadis and Dimitrios Soudris, editors. *Fine- and Coarse-Grain Reconfigurable Computing*. Springer, Berlin, 2007.

[VSL08] Frank Vahid, Greg Stitt, and Roman Lysecky. Warp Processing: Dynamic Translation of Binaries to FPGA Circuits. *Computer*, 41(7):40–46, 2008.

[WD03] Michael G. Wrighton and André DeHon. Hardware-assisted simulated annealing with application for fast FPGA placement. In *FPGA*, pages 33–42, 2003.

[WKS87] T. Watanabe, H. Kitazawa, and Y. Sugiyama. A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor. *IEEE Trans. CAD*, 6(2):241–250, 1987.

[Xil08] Xilinx Inc. EasyPath FPGAs. 2008.

[YL05] A.J. Yu and G.G.F. Lemieux. Defect Tolerant FPGA Switch Block and Connection Block with Fine-grain Redundancy for Yield Enhancement. In *Proceedings of 15th International Conference on Field Programmable Logic and Applications (FPL)*, pages 255–262, 2005.