Chapter 6
Concluding Remarks

In this book, methods architectures and tools have been proposed for runtime reconfigurable systems based on FPGAs. This results in substantial improvements in the efficiency of integrating reconfigurable modules into an FPGA, the tools to build such systems, and the capabilities to reconfigure a system at runtime. The key contributions are as follows:

Communication architecture for module-based on-FPGA communication: It has been proven that existing methodologies and communication architectures provide a tiling of the reconfigurable area that is too coarse-grained for most practical applications. With the novel ReCoBus and connection bar architecture, the size of the tiles is reduced by more than an order of magnitude while reducing the cost to implement the complete infrastructure and to increase throughput at the same time. This is the key technology improvement required to efficiently implement runtime reconfigurable systems.

Design flow and tools: For implementing the ReCoBus and connection bar communication architecture, a powerful tool has been developed that is capable to map the communication infrastructure in a regular structured manner on an FPGA. Furthermore, this tool provides a floorplanner capable to generate all location and resource constraints required to build the static system or the configurable modules.

Techniques for flexible high-speed runtime reconfiguration: For partial runtime reconfiguration, techniques have been proposed that allow to place modules in a two-dimensional fashion at high speed. In addition it was demonstrated how modules can be parameterized by configuration bitstream manipulation at runtime (e.g., setting a module base address). For a further acceleration of the configuration, additional techniques, including prefetching, configuration port overclocking, configuration decompression, and defragmenting the module layout have been examined.

With these architectures, methods, and tools, a technical basics for achieving a considerable benefit over completely statically implemented systems on FPGAs has been provided and demonstrated in different applications. This will help design

D. Koch, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*, Lecture Notes in Electrical Engineering 153, DOI 10.1007/978-1-4614-1225-0_6,

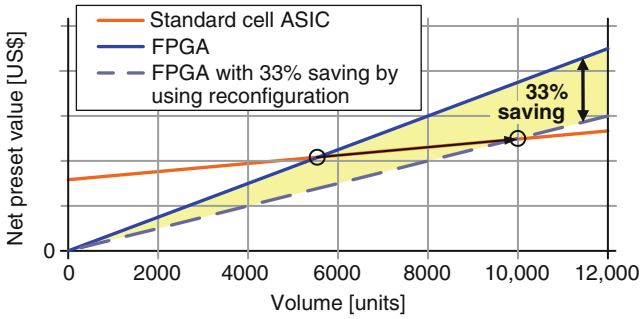

Fig. 6.1 Production cost of a hypothetical system over the units for an assumed ASIC or FPGA implementation (The values are taken from [Kot]). The FPGA has zero initial cost, but a much higher price per unit than compared with the ASIC solution. By utilizing partial runtime reconfiguration, the FPGA domain is shifted remarkable into higher volume regions, hence, squeezing the ASIC out of the mid range market

engineers in the field of reconfigurable computing to exploit the advantages of adapting systems at runtime. Additionally, the results are relevant for commercial applications and electronic design automation in general. With partial runtime reconfiguration, systems can be implemented cheaper and with lower power consumption by using smaller devices. Furthermore, this is an option to add additional functionality to existing systems. Even if only a relatively small part of the worldwide multi billion US$ FPGA market [EW0] can substantial benefit from runtime reconfiguration, the possible savings are many tens of millions US$ per year. However these savings will not decrease the total FPGA market. As illustrated in Fig. 6.1, the reduction of the FPGA unit cost – due to runtime reconfiguration – yields in a much wider range of designs that dominate an ASIC solution from an economical point of view. And the FPGA has further advantages, including faster time to market, possible after sales business by updates, lower economical risk due to the lower initial investment – just to mention a few of them. Moreover, runtime reconfiguration provides new opportunities in fast system start and system modularity. Consequently, partial runtime reconfiguration is advantageous for both, FPGA users and vendors.

A further important aspect is the fully encapsulated component-based design methodology that is advantageous for FPGA design and hardware development in general. With the here presented design flow, implemented and foremost fully placed and routed modules are composed together to complex systems based on interface specifications that also include the timing behavior. This enormously simplifies the system integration phase and addresses the demands of an easy design reuse to close the design productivity gap, as sketched in Fig. 6.2. With introducing design reuse, more functionality can be designed in less time and fully implemented modules will in particular ensure timing and reduce the module test expenses. Furthermore, the encapsulated design methodology prevents that changes in one part of the system will influence the rest of the system. Also the synthesis, and place and route itself will be accelerated as they can be carried out fully parallel for each module and

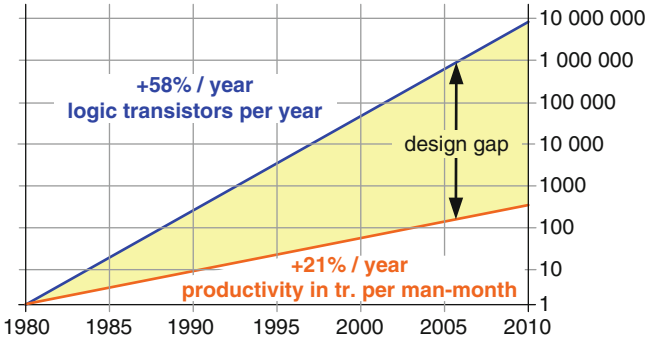

Fig. 6.2 Illustration of the design productivity gap. The progress in silicon process technology rises much stronger than the progress in electronic design automation (The values are taken from [Fly05])

because of the reduced problem size by partition the implementation steps in smaller units. In other words, this book provides the technical basics required for performing place and route in the cloud.

6.1 Future Directions

The range of applications and the field of research on runtime reconfigurable systems is huge; and in this book, focus was put on technical aspects required for efficient implementations of according FPGA-based systems. In this context, *efficiency* is used for both, the implementation process (supported by tools) as well as for the resulting systems (supported by architectures and methods). However, there remain challenging open problems for future research and development.

Most embedded systems integrate various hardware and software components. So far, the hardware/software partitioning (i.e., the decision, which part of the system will be implemented in hardware and which in software) is performed completely at design-time. As presented in Chap. 4, runtime reconfiguration – together with hardware/software morphing – provides interesting opportunities for self-adapting a system according to various objectives, including variable throughput demands, power consumption, or the currently available resources. However, the different hardware and software components have to communicate among each other. This demands compatible hardware/software interfaces that allow to link components that are arbitrary implemented in hardware or software (and that might change, even at runtime). This might also include a network layer with a dynamic binding of task among multiple nodes, as revealed for a ReCoNet. Here exists already good work that targets efficient [LK10, HHK10] interfaces or even complete message passing systems [GSS09, SPLC12, LP08]. It would be interesting to combine these approaches with partial reconfiguration and the dynamic hardware/software partitioning.

This could be further combined with modern design languages and behavioral compilers, such as Catapult-C from Mentor-Graphics or Impulse C from Impulse Accelerated Technologies. Such tools are not only helping in rising design productivity, they can automatically provide a range of implementation alternatives that are optimized for different objectives (typically resources versus throughput /latency). Consequently, various design alternatives for software as well as different hardware implementations may be automatically generated from the same specification. And these implementation might then be integrated on demand at runtime. This would combine the advantages of behavioral synthesis with the advanced system integration capabilities provided by the ReCoBus and connection bar communication architecture. Behavioral compilers allow communication constructs that may be automatically mapped to a customized or standardized communication architecture. For instance, pointers may result in a bus master interface to a ReCoBus and streaming ports may be mapped to corresponding connection bars.

Throughout this book, many issues that arise when implementing more advanced runtime reconfigurable systems have been presented. With tools like the RoCoBus-Builder, or the upcoming GOAHEAD (introduced in the appendix) most of these issues have been solved and automated such that a designer has not to care much about low-level details of the FPGA. In particular, this provides an easy migration path from one FPGA family to another one. However, there is still further room for automating the design of runtime reconfigurable systems. Here, it requires language extensions to specify specific properties of a reconfigurable system. Present RTL HDL languages are made for completely static ASIC designs and do not include constructs to express dynamic aspects, as they might arise for FPGAs. For example there are no statements to express that modules are executed mutually exclusive to each other (which means that they can share FPGA resources), or that a module might be preempted. There are also no nice ways to describe reconfiguration sequences of modules. Moreover, simple data flow models describe mainly an order of modules rather than a precise communication behavior. For instance, it is a big difference if a module is pipelined in the sense that it consumes an input value and produces an output value in, lets say, every clock cycle (e.g., a FIR filter) or if a module produces only a single result after some complex processing (e.g., a CRC checker). Consequently, the temporal module placement example of a DFG in Fig. 1.20 on Page 37 has to be refined to automatically derive an optimized module packing and communication synthesis. In this field exist already good entry points. Besides the system level initiatives, there are very promising approaches using polymorphism in object oriented languages to express reconfiguration [SNH+10, Abe11].

We can conclude, that runtime reconfiguration is ready to use and that it will be used in many upcoming systems order to meat performance, cost, or power requirements.