Reconfigurable Architectures: A Survey of Design and Implementation Methods

Thijs van As (1143840), Siebe Krijgsman (1153463)
{T.vanAs, S.Krijgsman}@student.tudelft.nl

Advanced Embedded Architectures, ET4 360
Department of Computer Engineering
Faculty of Electrical Engineering, Mathematics & Computer Science
Delft University of Technology

Abstract

Reconfigurable architectures are becoming increasingly popular among designers. This paper presents a survey of design and implementation methods of reconfigurable architectures. Among basic implementation characteristics, system-level architectures are presented. Differences between fine- and coarse-grain development are becoming more important. Current trends in reconfigurable computing are adding more heterogeneous functions to the hardware, using soft-core processors and using coarse-grained fabrics.

1 Introduction

Designing architectures for (embedded) computer systems using reconfigurable hardware is becoming more popular now that classical drawbacks are diminishing. Hardware like field programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs) are used as building blocks for reconfigurable computing.

In this paper, a survey of reconfigurable architectures is presented. First, some background information about reconfigurable computing is provided. After this, different system concepts will be discussed. After evaluating fine-grain and coarse-grain design methods, trade-offs are given between those two.

The remainder of this paper is organized as follows. Section 2 provides basic insight in reconfigurable computing, as well as reasons why to choose for it. In Section 3, characteristics of FPGAs and CPLDs are discussed. Subsequently, Section 4 presents an overview of system-level architectures, and how they are used by today's industry. Section 5 discusses different methods of reconfiguring devices. Different aspects of fine-grain and coarse-grain design are evaluated in Section 6. These aspects are followed by design trade-offs in Section 7. Section 8 presents the conclusions of the survey.

2 Background

Reconfigurable computing is getting more and more important in the (embedded) computing world. This is mostly because of the speed/flexibility trade-off which holds in architectural design. This Section gives more insight about the background of reconfigurable computing.

2.1 Speed/flexibility trade-off

When designing computer systems, one faces the speed/flexibility trade-off at some point [16].

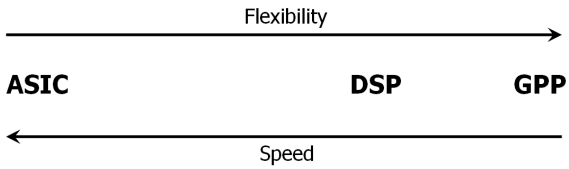

Figure 1: Speed/flexibility trade-off

The designer has to make a choice whether the design focus will be on the speed of the system, or on the functional flexibility. Figure 1 presents a schematic overview of the speed/flexibility trade-off.

The general purpose processor (GPP) stands at one end of the spectrum. A GPP is very flexible, as it can execute any function. Each function can be assembled by various instructions supported by the GPP. However, fetching instructions from memory and decoding them costs (a lot of) time. This, combined with the fact that most (complex) functions are assembled of (a lot) more than one instruction, explains why a GPP is relatively slow.

At the other end of the spectrum, an application specific integrated circuit (ASIC) can be found. Because an ASIC is designed for one specific task, there is no need for an instruction set. This implies that the design of an ASIC can be optimized for this specific task, and can thus execute it relatively fast.

Solutions at other places in the spectrum, generally consist of digital signal processors (DSPs). It should be noted that the exact position of a DSP in the spectrum cannot be given: this depends on the type of DSP. Some DSPs are designed to be more flexible; others to have a higher performance.

2.2 Why choose for reconfigurable computing?

With the use of reconfigurable hardware, one tries to fill the gap between a hardware-only (ASIC) and a software-only (GPP) solution [8]. It is intended to achieve a higher performance than a software-only solution, and maintain more flexibility than a hardware-only solution.

Research results by Stitt et al [18] show that moving critical software loops to reconfigurable hardware results in an average speedup of 3 to 7 times compared to the original system. In addition to this speedup, energy savings of 35% to 70% are reached. These are good motivations to choose for a reconfigurable architecture.

2.3 Granularity

An aspect which is very important in reconfigurable design, is *granularity*. The granularity denotes the hardware abstraction level [16][15]. Generally speaking, there are two forms of granularity: fine-grained reconfiguration, and coarse-grained reconfiguration.

When a fine-grained approach is used, it is possible to manipulate bitwise, so every single bit can be separately routed/used. Configuration is usually done by means of a netlist, which describes the connections within an electronic circuit.

A coarse-grained design, in contrast, usually does not allow bitwise manipulation. Word or sub-word maniplulation is more generally the case in coarse-grain design. Coarse-grain designs are usually configured by means of connections between complete functional units.

3 Design approaches

The most used reconfigurable logic devices in industry are FPGAs [8][19][5] and CPLDs [5]. This Section presents differences between these devices, and implementation details.

3.1 FPGAs and CPLDs

A big practical difference between FPGAs and CPLDs is that an FPGA is volatile, and a CPLD is non-volatile. This means that an FPGA cannot hold its configuration when powered off, but a CPLD can.

An FPGA is based on configurable logic blocks (CLBs). One CLB is based on one or

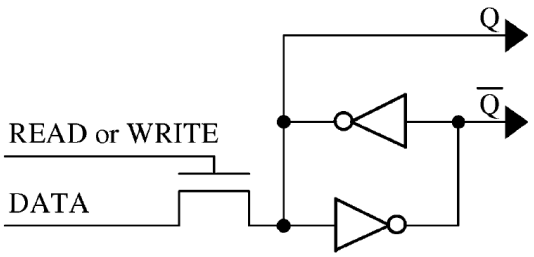

Figure 2: SRAM bit

more lookup tables (LUTs). In the next Section, the implementation of a CLB is presented. In Section 3.3, the implementation of a LUT is provided. There are two classical ways of configuring FPGAs. The first one is based on antifuses, the second one is based on SRAM memory.

An antifuse is the opposite of a normal fuse; when the voltage across the antifuse exceeds a certain level, a connection is made. The antifuse technology is not very popular these days, because a programmed antifuse FPGA is not reconfigurable.

SRAM [6][7] stands for static random access memory. It should be noted that this is in most cases not a good name; only in a few FPGAs the SRAM memory is randomly accessible. SRAM memory holds its state only when powered on. Figure 2 shows the implementation of an SRAM bit.

A CPLD is based on logic array blocks (LABs). These LABs consist of macrocells, which are AND/OR planes. Because they are using AND/OR planes, electrically erasable programmable random access memory (EEPROM) can be used for configuring CPLDs. An advantage of this method is that the state of memory is kept after power-off.

In an FPGA, EEPROM memory cannot be used for configuration, because the type of switch that is needed is one that can simply connect or isolate two wires, rather than combining wires in a wired-AND or wired-OR fashion [5].

FPGA based architectures are considered fine-grain, and CPLD architectures coarse-grain. This

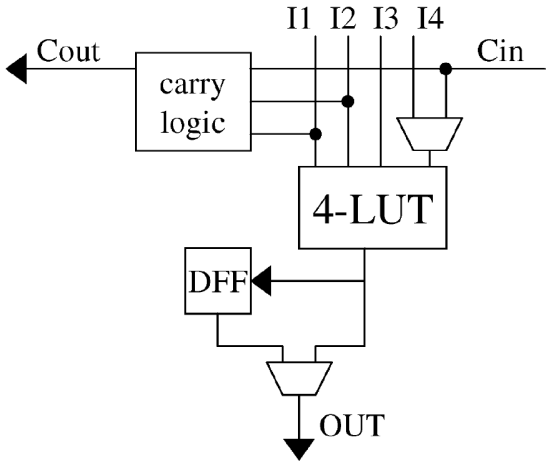

Figure 3: Standard CLB design

is because a CPLD is configured using (large) macrocells. Of course the granularity is a point of discussion in this context. Section 6 will elaborate more on this topic.

3.2 Implementation of a CLB

Figure 3 shows an example of a CLB design. Here, a 4-input LUT is used for determining the CLB function. Some carry logic is added to gain performance in additions and large AND-operations. A D flip-flop (DFF) is included to provide stateholding elements. This DFF can be bypassed when not needed, by selecting the appropriate multiplexer input behind the flip-flop.

3.3 Implementation of a LUT

Most commercial FPGAs contain three to six input LUTs. In Figure 4, a three-input LUT is shown. In principle, a LUT is a truth table for an arbitrary function. With an N-input LUT, 2^N configuration bits have to be programmed. So any N-input logical function can be described. Every combination of input bits corresponds to exactly one programmed bit.

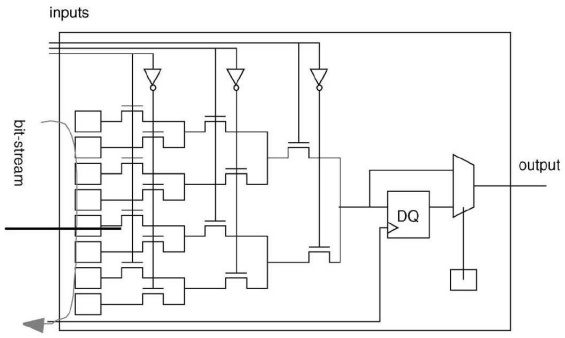

Figure 4: Three-input LUT design

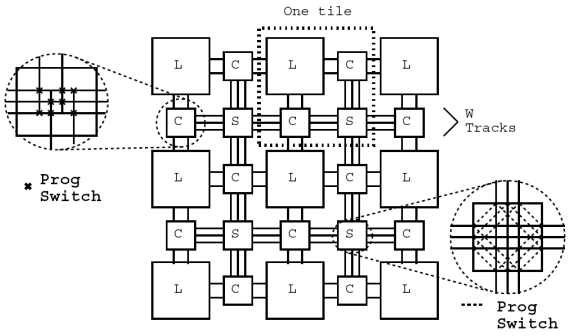

Figure 5: Routing switchboxes

3.4 Switchbox routing

Figure 5 depicts an FPGA architecture. Blocks with an L inside represent logic blocks, blocks with a C are connection blocks, and blocks with an S are switch blocks. The difference between connection blocks and switch blocks is that connection blocks make connections between logic blocks, and switch blocks between channels. Figure 6 shows how a connection is made in a switch block with an SRAM bit like in Figure 2.

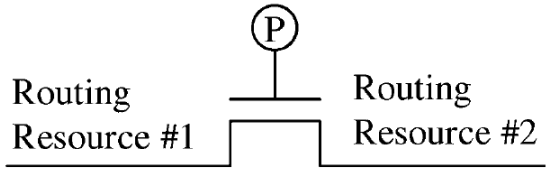

Figure 6: SRAM bit used for routing

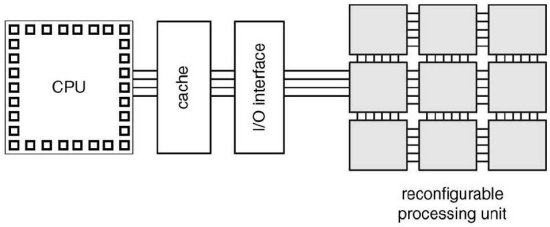

Figure 7: Architecture I - External processing unit

4 System-level architectures

This Section discusses different reconfigurable system-level architectures. Architectures I to IV are adapted from [8], architecture V is adapted from [19]. Each of the following Sections presents one architecture.

4.1 I External processing unit

Figure 7 shows the first architecture. The architecture consists of a central processing unit (CPU), with its own cache memory and I/O interface, and a separate reconfigurable processing unit. The reconfigurable unit communicates with the CPU using the I/O facilities of the CPU. The unit acts like a normal peripheral to the CPU. Because of this, the communication costs are relatively high. Communication is not optimized for high performance calculations. A system with this architecture is useful in situations where communication between the CPU and the reconfigurable unit is not needed continually. A suitable application for this architecture is for example emulation.

4.2 II Attached processing unit

The second architecture is shown in Figure 8. The architecture consists of a CPU with its own cache memory and a reconfigurable unit connected to the outside world using the same I/O interface. The infrastructure on how the CPU and the reconfigurable unit communicate is similar those on symmetric multiprocessing (SMP) computers. Both units have their own cache memory,

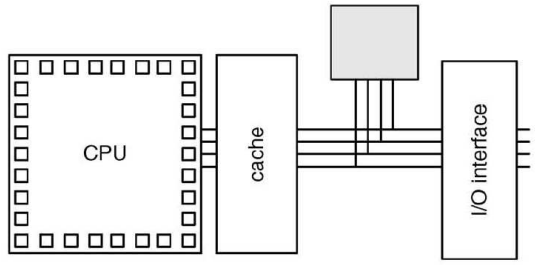

Figure 8: Architecture II - Attached processing unit

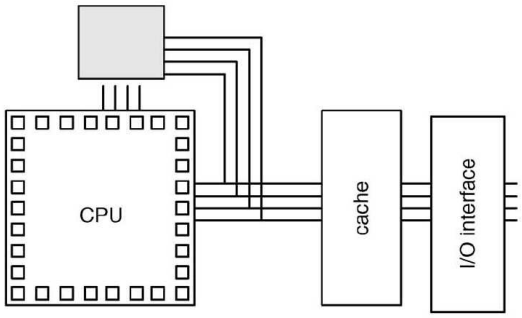

Figure 9: Architecture III - Co-processor

but they share the I/O interface. An advantage of this approach is that the communication costs between the two units is much lower than in the first architecture.

4.3 III Co-processor

The co-processor design is shown in Figure 9. The reconfigurable unit acts as a co-processor for the CPU. They have both access to the same cache memory, and have the same connection to the outside world. Because the reconfigurable unit is attached to the CPU itself, communication costs are lower than in the first two architectures.

4.4 IV Reconfigurable functional unit

Figure 10 depicts the fourth architecture. In this architecture, the reconfigurable hardware is used to provide reconfigurable functional units within a host processor. An advantage of this architecture is that programming can be done by us-

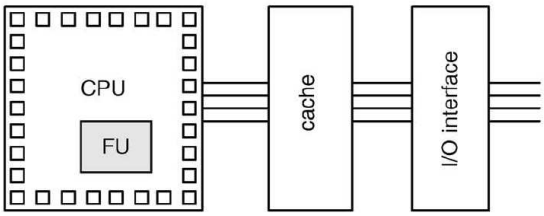

Figure 10: Architecture IV - Reconfigurable functional unit

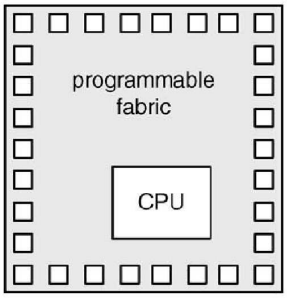

Figure 11: Architecture V - Embedded processor

ing an already established programming environment. To be added are custom instructions, that might change over time. The reconfigurable units execute as functional units on the main microprocessor datapath.

4.5 V Embedded processor

The fifth architecture, depicted in Figure 11, looks like an inverted version of the fourth architecture. In this situation, a CPU is embedded inside the reconfigurable fabric. It is possible to subdivide the CPU part into some variants:

- Heterogeneous functions
- GPP cores
  - Hard-wired core
  - Soft-core

Heterogeneous functions are hard-wired logical functions inside a reconfigurable device. Examples of such functions are (large) multipliers, dividers, shifters or registers. It is a trend to see

more of these functions in FPGAs today, and thus make them more coarse-grain architectures.

GPP cores can reside in two forms inside the reconfigurable fabric. They can either be hard-wired, or they can be a so called soft-core [17]. A hard-wired core is usually an established core, for which a complete tool-chain (compiler, linker, debugger, etc) is available. A soft-core is a processing core which resides in the reconfigurable part of the hardware. It can be very easily adapted to the wishes of the designer, by adding or removing peripherals.

For example, the Xilinx Virtex II-Pro [3] FPGA embeds two hard-wired IBM PowerPC [2] cores. As for soft-cores, the Xilinx MicroBlaze [3] and Altera NIOS II [1] RISC cores are much used nowadays.

5 Reconfiguration methods

For fine-grained reconfigurable designs two reconfiguration schemes are available. These two models are *compile-time reconfiguration* and *run-time reconfiguration* [16].

5.1 Compile-time reconfiguration (CTR)

In this scheme the hardware is reconfigured before the start of operation. It can no longer be changed once the hardware is in operation. This method is the most straight forward, however it poses some constraints to the size and the flexibility of the design [16].

5.2 Run-time reconfiguration (RTR)

In comparison with CTR, RTR enhances the functional density of a design. Functional density is modeled as

$$D = 1/AT$$

with D being the functional density, A the function area and T the execution time. In order to increase the functional density, some problems specific to RTR have to be overcome.

The first problem that arises is the *temporal partitioning* of a design. A design has to be partitioned into time-exclusive segments, and specific hardware has to be designed for each segment. Most application break down into segments quite naturally [16]. Secondly, *reconfiguration overhead* has to be considered. In order to achieve high clock rates, the time used to reconfigure the hardware inbetween phases has to be kept to a minimum. The last problem that has to be addressed is *interconfiguration communication*. As data from each segment will be used for (one of the) next segments, this data needs to be kept in memory that will be sustained inbetween the different reconfiguration phases.

6 Fine- and coarse-grain design

This Section presents characteristics of fine- and coarse-grain design.

6.1 Basics of fine-grain design

When considering fine-grain design, FPGAs are the most common example of the design approach. When implementing an application on fine-grain level, there are 3 main parameters that have to be considered, the first of which is the *block granularity*. As there are numerous FPGAs on the market, it is important to know the number of basic elements (usually LUTs, multiplexors and/or flipflops) in one logic block. The *density*, measured in either number of logic blocks or number of equivalent gates, is the second important design parameter. The last parameter is the *reconfiguration time*. In a design that uses CTR, this parameter is not of the utmost importance, but in order to make the use of RTR efficient, the reconfiguration time should be kept to a minimum. Next to total reconfiguration, partial reconfiguration can also be achieved.

6.2 Basics of coarse-grain design

In order to enable coarse-grain design on a fine-grain architecture (FPGA), some design innova-

tion is required. To make use of coarse-grain routing, groups of single-bit wires have to be routed, making use of a single configuration bit (Figure 5-b) [19]. This poses some constraints to the routing flexibility of the design. Also, basic logic blocks of the FPGA have to be grouped into clusters to make the use of the coarse-grain routing scheme efficient. Figure 17 [20] depicts such a situation. The trade-offs for the loss of flexibility in this design are a reduction in power, area, delay and configuration time. In order to make these reductions, a new CAD tool needs to be introduced [12]. The second approach towards coarse-grain design is the use of coarse-grain arrays. Due to the large reconfiguration overhead in FPGAs and, partially overlapping, the high wiring/logic ratio, more coarse-grain solutions are being developed. One of them is the *KressArray* [13]. The KressArray is a lot less overhead-prone and more area efficient than FPGAs (see Figure 12 [4]). The KressArray consists of Processing Elements called rDPU (reconfigurable DataPath Unit) arranged in a NEWS (North, East, West, South) network [14]. Figure 13 shows a KressArray with 9 rDPUs. Figure 14-a shows the layout of one rDPU with all its connection, Figure 15-b shows all possible east-west connections, Figure 14-c all possible north-south connections. Figure 15-d is an rDPU serving as an arithmetic operator, Figure 15-e as an arithmetic/routing unit and Figure 15-f as an exclusive routing operator.

6.3 Implementation of coarse-grain routing on an FPGA

When coarse-grain routing and logic block clustering are used, the following implementation could be implemented on FPGA. As shown in Figure 18, the clusters are again clustered in super clusters. In this implementation both coarse-grain tracks and fine-grain tracks are used to find an optimal solution in the fine/coarse track ratio. The data-path granularity (the number of clusters per super-cluster) is set to 4, which is proven to be a good choice in [11]. The cluster size, i.e. the

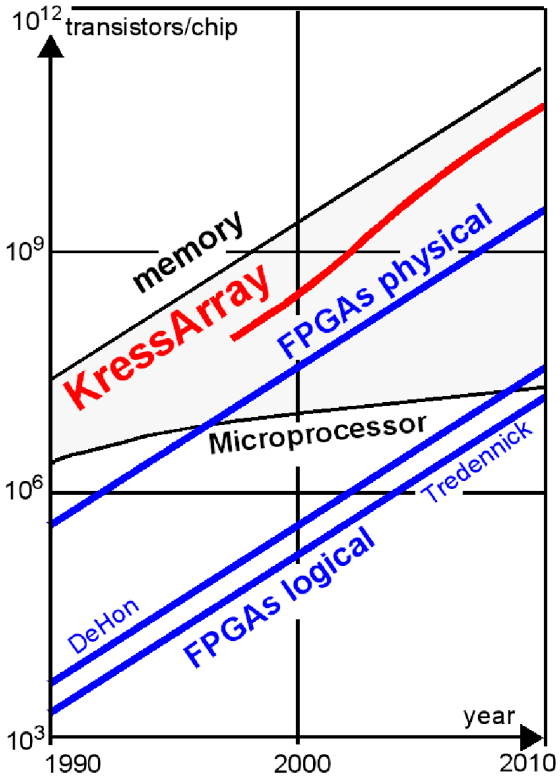

Figure 12: Gordon Moore curve for several devices

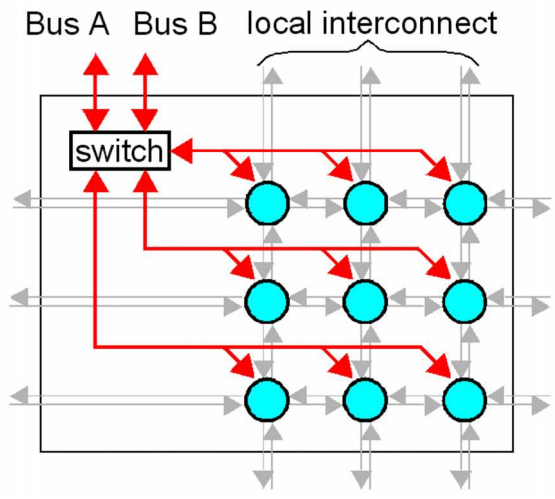

Figure 13: KressArray with 9 rDPUs

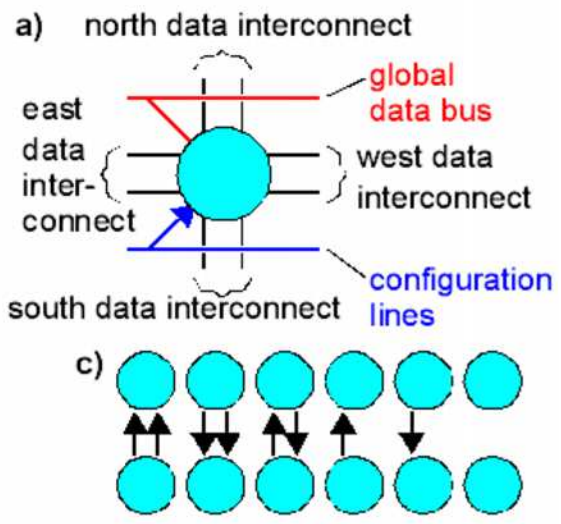

Figure 14: rDPU connections

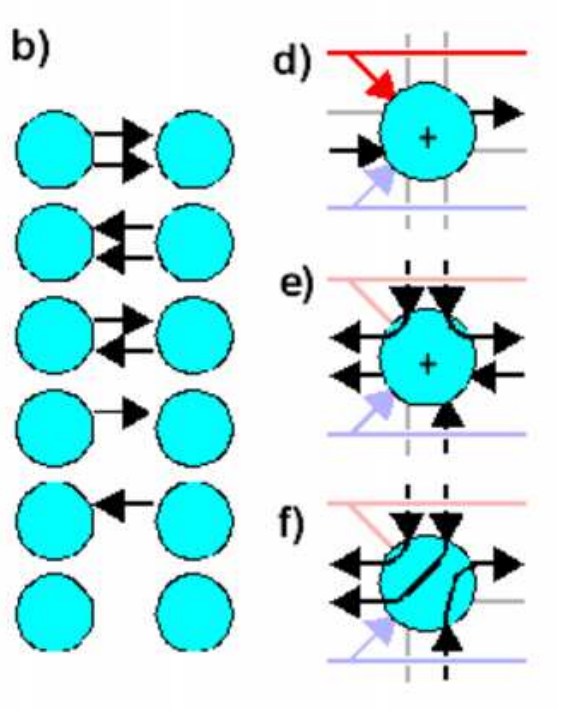

Figure 15: rDPU connections

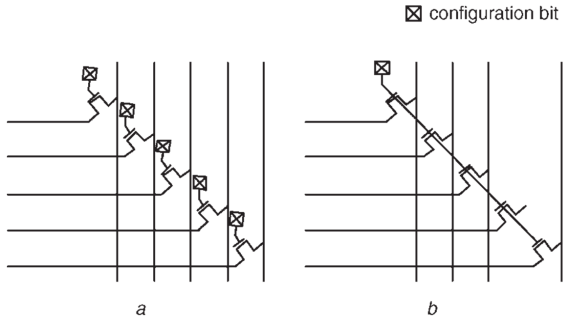

Figure 16: Routing architectures

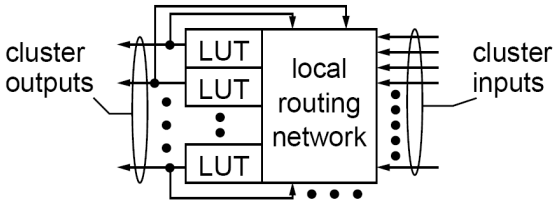

Figure 17: Logic cluster

number of logic blocks per cluster, is also set to 4, as discussed to be a good choice in [9].

In Figures 19, 20 and 21 [20] the input, output and switch blocks are shown, respectively. It clearly shows that both fine- and coarse-grain tracks have been mixed.

6.4 The gain of coarse-grain routing

In Figure 22 [20] a plot is shown of the percentage of coarse-grain tracks in the design, where 0% is full fine-grain. It can be seen that the inflexibility introduced by a small number of coarse-grain tracks only introduces more area. However, as a larger part of the tracks is used for coarse-grain routing, the advantages start to

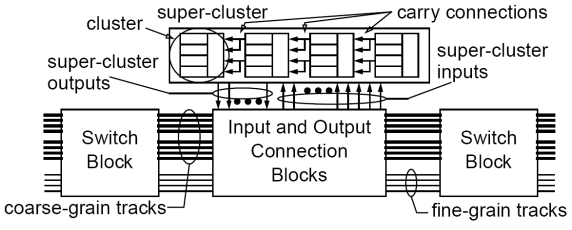

Figure 18: Super cluster topology

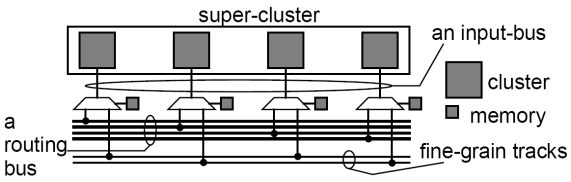

Figure 19: Input connection block

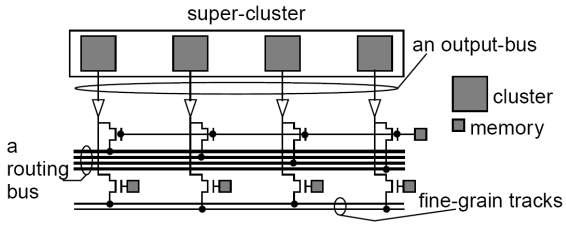

Figure 20: Output connection block

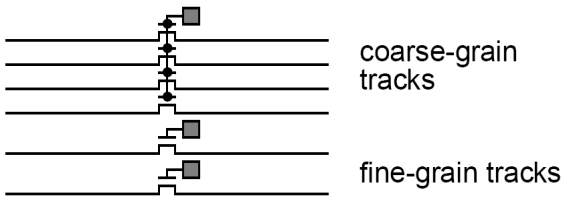

Figure 21: Switch block

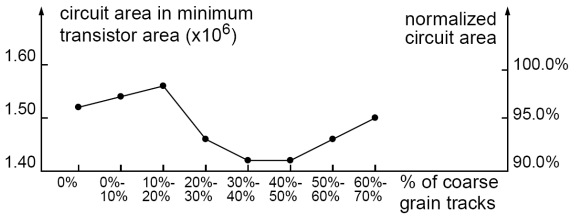

Figure 22: Area/performance trade-off

pay off, with a clear optimum between 30% and 50%.

7 Fine-grain VS coarse-grain trade-offs

As mentioned earlier, in order to use coarse-grain routing on an FPGA, a new CAD tool has to be designed, adding significantly to the development time [12]. Upon introduction, the routing of coarse-grain tracks will diminish the flexibility in routing. More broad tracks will enlarge this problem [20]. According to [10], wire delay will keep increasing in comparison to the functional unit delay. Therefore coarse-grain design will become increasingly attractive in the future [16].

8 Conclusions

Currently, the trends in reconfigurable computing are threefold. Firstly, it is becoming more popular to implement more heterogeneous functions to speed up more complex calculations. Secondly, soft-cores are included in reconfigurable devices to increase the flexibility and provide support for general purpose computing and programming. Finally, the use of coarse-grained fabrics and coarse-grain designs are becoming a topic of increasing interest in this field [19]. Especially as the wire delays keep increasing relative to the functional unit delay, the use of coarse-grain designs will become more common [16][10].

As more RTR challenges are overcome and the reconfiguration times decrease, more appli-

cations for this strategy arise [16].

References

[1] *Altera*. http://www.altera.com/.

[2] *IBM PowerPC*. http://www.ibm.com/chips/power/powerpc/.

[3] *Xilinx*. http://www.xilinx.com/.

[4] J. Becker, R. Hartenstein, M. Herz, and U. Nageldinger. Parallelization in co-compilation for configurable accelerators. *Proceedings of Asia and South Pacific Design Automation Conference, ASP-DAC '98*, February 1998.

[5] S.D. Brown. An overview of technology, architecture and cad tools for programmable logic devices. *IEEE 1994 Custom Integrated Circuits Conference*, pages 69 – 76, 1994.

[6] P. Chow, S. Ong Seo, J. Rose, K. Chung, G. Paez-Monzon, and I. Rahardja. The design of an sram-based field-programmable gate arra - part i: Architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(2):191 – 197, June 1999.

[7] P. Chow, S. Ong Seo, J. Rose, K. Chung, G. Paez-Monzon, and I. Rahardja. The design of an sram-based field-programmable gate arra - part ii: Circuit design and layout. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 101 – 110, 1999.

[8] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171 – 210, June 2002.

[9] A. Ye et al. Fpga architecture for datapath circuits, 2003.

[10] D. Burger et al. Billion-transistor architectures, 1997.

[11] D. Cherepacha et al. Dp-fpga: an fpga architecture optimized for datapaths, 1996.

[12] V. Betz et al. Architecture and cad for deep-submicron fpgas, 1999.

[13] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger. On reconifugarble co-processing units. *Proceedings of Reconfigurable Architectures Workshop (RAW98)*, March 1998.

[14] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger. Using the kressarray for reconfigurable computing. *Proceedings of SPIE*, 3526, November 1998.

[15] W.H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V.K. Prasanna, and H.A.E. Spaanenburg. Seeking solutions in configurable computing. *IEEE*, pages 38 – 43, December 1997.

[16] M. Platzner. *Reconfigurable Computer Architectures - Rekonfigurierbare Rechnerarchitekturen*. Stanford University, 1998.

[17] J. Rose. Hard vs. soft: The central question of pre-fabricated silicon. *IEEE Procedeings of the 34th International Symposium on Multiple-Valued Logic*, 2004.

[18] G. Stitt, F. Vahid, and S. Nematbakhsh. Energy savings and speedups from partitioning critical software loops to hardware in embedded systems. *ACM Trans. Embedded Comput. Syst.*, 3(1):128 – 232, March 2004.

[19] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, and P.Y.K. Cheung. Reconfigurable computing: architectures and design methods. *IEE Proc.-Digit. Tech.*, 152(2):193 – 207, March 2005.

[20] A. Ye, J. Rose, and D. Lewis. Architecture of datapath-oriented coarse-grain logic and routing for fpgas. *IEEE 2003 Custom Integrated Circuits Conference*, pages 61 – 64, 2003.