High-Level Estimation and Trade-Off Analysis for Adaptive Real-Time Systems

Ingo Sander, Jun Zhu, Axel Jantsch
Royal Institute of Technology
Stockholm, Sweden
{ingo, junz, axel}@kth.se

Andreas Herrholz†, Philipp A. Hartmann†, Wolfgang Nebel‡
†OFFIS Institute, ‡Carl v. Ossietzky University
Oldenburg, Germany
{herrholz,hartmann,nebel}@offis.de

Abstract

We propose a novel design estimation method for adaptive streaming applications to be implemented on a partially reconfigurable FPGA. Based on experimental results we enable accurate design cost estimates at an early design stage. Given the size and computation time of a set of configurations, which can be derived through logic synthesis, our method gives estimates for configuration parameters, such as bitstream sizes, computation and reconfiguration times. To fulfil the system's throughput requirements, the required FIFO buffer sizes are then calculated using a hybrid analysis approach based on integer linear programming and simulation. Finally, we are able to calculate the total design cost as the sum of the costs for the FPGA area, the required configuration memory and the FIFO buffers. We demonstrate our method by analysing non-obvious trade-offs for a static and dynamic implementation of adaptivity.

1. Introduction

This paper proposes a novel estimation method to be able to estimate the design costs for the implementation of adaptive applications with given throughput constraints on reconfigurable architectures at an early stage of the design process. In particular we use our estimation method to analyse non-obvious trade-offs between a dynamic and a static implementation of an adaptive streaming application.

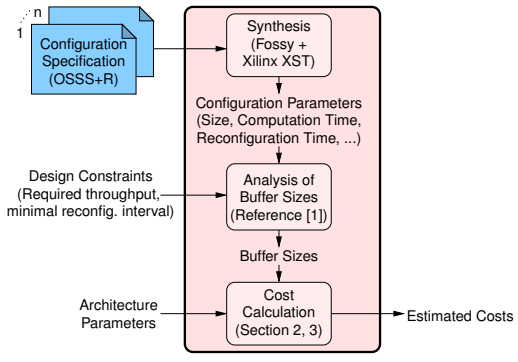

Figure 1. Design Estimation Method

Our estimation method, which should be seen as a part of a larger design exploration flow, is illustrated in Figure 1. The estimation method assumes that there are several configurations that are never to be executed simultaneously and thus can share a reconfigurable area on an FPGA. Logic synthesis yields configuration parameters, which together with design constraints form the input for an automated analysis to dimension the buffer requirements for a given output data rate. Our performance analysis method uses a hybrid approach based on integer linear programming and simulation and is elaborated in [1]. The obtained buffer sizes are then used together with architecture parameters for the calculation of the design costs. This allows to estimate the design costs before using explicit back-end tools for partial reconfiguration, like the Xilinx EAPR flow [2].

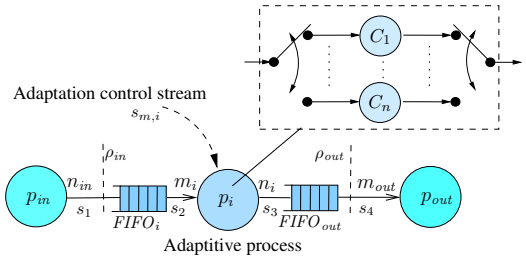

Figure 2. Adaptive streaming application

To dimension the buffer sizes we model our target applications using the adaptive streaming application model illustrated in Figure 2. Nodes denote the computation processes. Edges associated with FIFOs denote the communication channels with finite storage, which decouple the input data streams from output data streams of each communication channel (e.g. $FIFO_i$ decouples s_1 from s_2). Processes read tokens from the input-side FIFOs, and emit the produced data tokens to the output-side FIFOs at the end of the computation. The input/output token numbers are fixed [3] and denoted as symbols at each side of the communication channels, e.g. process p_i has m_i input tokens and n_i output tokens. Meanwhile, the adaptive process p_i responds to the adaptation control stream $s_{m,i}$, and can switch between n different working configurations C_1 to C_n, as shown in the dashed box. While the stream source p_{in} provides a peak data rate ρ_{in}, an average output data rate ρ_{out} needs to be guaranteed by the application even during the run-time reconfiguration of process p_i. The adaptation control signal $s_{m,i}$ might either come from an external controller or be

retrieved from the data streams. Given the reconfiguration and computation times for each configuration and a worst case reconfiguration interval, our analysis method is able to calculate the buffer sizes to guarantee the required average data rate at the output as elaborated in [1].

Since the description of dynamically reconfigurable systems is not explicitly supported by traditional hardware description languages, we have chosen to combine our performance analysis method with OSSS+R [4], a methodology for modelling, simulation and synthesis of reconfigurable systems based on SystemC. In combination with the synthesis tool *Fossy* [5], OSSS+R provides a direct path to an implementation on FPGA-based platforms. Several approaches to practical modelling of reconfigurable systems are known, some of which cover synthesis as well, like [6], [7]. They propose a design flow leading to a real implementation. However, design efficiency of adaptive real-time embedded systems has not been addressed. In this work, we do not consider configuration scheduling [8], since we assume that the system has to fulfil certain requirements, regardless of any required reconfigurations. The dynamic approach presented in [9] may be combined with our approach, to obtain the required minimal period between two reconfigurations.

2. Abstract Architecture Model

In the following we present two abstract architecture models that allow to compare the design costs of a dynamic implementation of adaptivity (just-in-time adaptivity) and a static implementation (mode adaptivity) at an early design stage.

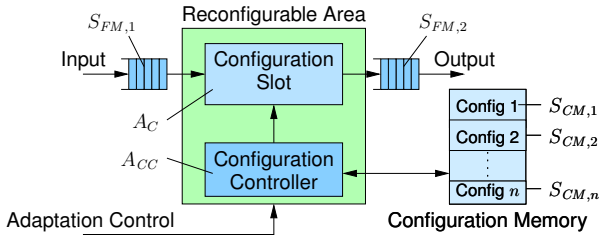

Figure 3. Model of just-in-time adaptivity

Figure 3 shows an abstract model of the architecture for *just-in-time adaptivity* (JIT). Just-in time adaptivity uses the capabilities of the reconfigurable FPGA with the objective to reduce the area requirements by means of a single reconfigurable slot. Every time a system function is needed, the function is loaded into this reconfiguration slot. All available configurations are stored in a configuration memory.

In order to abstract from technology details and to obtain a simple, but general and sufficiently accurate model, we make the following assumptions:

- The only unit we use for area cost is the number of logic elements (LE). The cost for a configuration i stored in the configuration memory is $A_{CM,i} = k_{CM} S_{CM,i}$, where k_{CM} expresses the relative memory cost depending on the technology, and $S_{CM,i}$ is the size of the configuration in bytes. The cost for a FIFO buffer i has to be described by a function $A_{FM}(S_{FM,i})$, since it does not only depend on the size $S_{FM,i}$, but also on the implementation of the FIFO buffer.
- The configuration slot A_C has at least the size of the largest configuration.
- The reconfiguration time $t_{R,i}$ is technology dependent, which is expressed by the factor k_R. It grows linearly with the size of the bitstream $S_{CM,i}$ of the configuration i, thus $t_{R,i} = k_R S_{CM,i}$.

Then, the total cost for just-in-time adaptivity is

$$A_{JIT} = A_{CC} + \sum_{i=1}^{n} k_{CM} S_{CM,i} + A_C + A_{FM}(S_{FM,1}) + A_{FM}(S_{FM,2}) \quad (1)$$

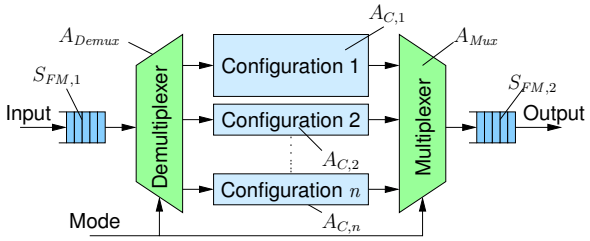

Figure 4. Model of mode adaptivity

Figure 4 shows a direct implementation of *mode adaptivity*. All system functions are statically available from start, which means that the system does not require the reconfiguration infrastructure, like the configuration memory and controller. Depending on the mode the corresponding system function is selected.

The total area cost for mode adaptivity is calculated as the sum of the area of all configurations, the area needed for multiplexer and demultiplexer, and the costs for the FIFO buffers.

$$A_{Mode} = \sum_{i=1}^{n} A_{C,i} + A_{Mux} + A_{Demux} + A_{FM}(S_{FM,1}) + A_{FM}(S_{FM,2}) \quad (2)$$

We can conclude that each of the presented alternatives has its strength and weaknesses. Mode adaptivity is very fast, but may result in a large area. Just-in-time adaptivity saves area due to the possibility to store configurations in compressed way in a memory, while they are not executed. However, additional buffers have to be used, and due to reconfiguration the maximum throughput may be lower than in a mode adaptive implementation. As our experiments show in Section 4 the trade-off between just-in-time and mode adaptivity is far from obvious and depends on the concrete application.

3. Platform Characterisation

Before our estimation method can be applied to a specific target architecture, it has to be calibrated, i.e. the parameters of the abstract reconfiguration architecture defined in Section 2 have to be replaced with specific values. For this paper, we have chosen OSSS+R [4] and the Xilinx Virtex-4 architecture for demonstration and validation of our flow. In OSSS+R reconfigurable areas of an FPGA are represented by so called *reconfigurable objects*. Configurations are modelled by means of different classes which can be mapped to one or more of these objects. Using the synthesis tool *Fossy*, an OSSS+R model can be automatically synthesised to register transfer level. The generated VHDL can be further processed by third-party gate-level synthesis tools and platform specific flows for reconfigurable systems.

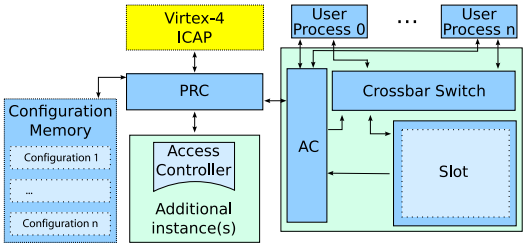

Figure 5. Synthesised OSSS+R Architecture

In Figure 5 a block-diagram of the resulting RTL architecture of an OSSS+R model is shown. For each reconfigurable object a corresponding *slot* is generated. The configurations of a slot are generated as a stand-alone module, which are later implemented as partial bitstream. Each slot is managed by an *Access Controller* (AC), controlling access to the slot and requesting reconfigurations. Reconfiguration requests are directed to the *Platform Reconfiguration Controller* (PRC). The PRC arbitrates and controls accesses to the platform specific reconfiguration port, like the internal configuration access port (ICAP) for Virtex FPGAs.

To characterise our analysis for OSSS+R and its *Fossy* based implementation flow we have implemented the adaptive streaming application given in Figure 2 as a parametrisable OSSS+R design, consisting of one reconfigurable object and accompanying in- and output FIFOs. We implemented an adaptive polynomial evaluator with varying order and parallelism to obtain configurations with different computation times and sizes. The type and number of configurations can be varied to measure changes in costs for reconfiguration infrastructure and in bitstream sizes. As target architecture we have used a Xilinx Virtex-4 LX25. One slice of the FPGA corresponds to one logic element (LE) in our methodology.

As a result of our experiments, we have (1) found out that the area cost A_{CC} of the controller for just-in-time adaptivity is almost independent of the number of configurations, (2) determined the cost for Block RAM $A_{FM(BM)}$ and Distributed RAM $A_{FM(DM)}$ for a given FIFO depth, (3) determined the costs for the multiplexer A_{Mux} and demultiplexer A_{Demux} for a given number of configurations, and (4) approximated the size of the configuration bitstream $S_{CM,i}$ for a given configuration size $A_{CM,i}$ and a given size of the configuration slot A_C. These parameters are used in the experiments in Section 4.

4. Trade-Off Analysis

We have conducted several experiments using the flow of Figure 1 to be able to analyse the trade-off between just-in-time and mode adaptivity. For these experiments, we have used the buffer dimensioning method of [1], the abstract models of Section 2, and the architecture parameters we have derived using the results of Section 3.

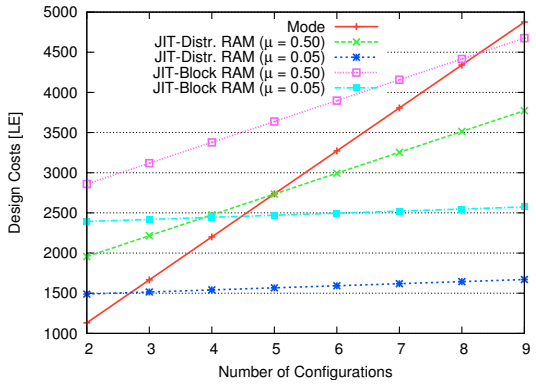

Figure 6. Costs for varying number of configurations

Figure 6 illustrates the trade-off between just-in-time and mode adaptivity for a varying number of configurations of almost identical size. In this experiment logic synthesis for six different polynomials of the same order gave a configuration size $A_{C,i} = 504 \pm 15$ LE and a constant computation time $t_{C,i} = 2.62\mu$s. For the following calculation we assumed $A_{C,i} = 519$ LE for all configurations and set A_C = 675 LE as configurable slot area. We then calculated the bitstream size $S_{CM,i} = 75\,085$ bytes and reconfiguration time $t_{R,i} = 751\mu$s. As design constraints, the required throughput was specified as $\rho_{out} = 3.05$ Mbit/s for a worst case reconfiguration interval of $t_{interR,i} = 1.57ms$. Our buffer analysis method of [1] returned that the buffer sizes required to meet the throughput constraints are a minimal buffer depth of 1 for the input and 72 for the output FIFO, which is independent of the number of configurations.

It is in practice impossible to give a general relation for the memory cost factor, since it not only depends on the cost of the configuration memory, but also on the cost of the logic elements in an FPGA and the control logic that is required to connect the memory to the FPGA. An SRAM with a simpler memory interface can be about 50–100 times more expensive than a DRAM of the same size. Thus we

assume that k_{CM} is specified by the designer who has a profound knowledge of the architecture. We have conducted the experiments with different values $k_{CM} = \mu A_{C_i}/S_{CM,i}$, where a factor $\mu = 0.5$ means that the cost for storing a configuration in the memory is 50% of the cost when implementing the configuration in the logic elements of the FPGA. We found that the values $k_{CM} = 0.5$ and $k_{CM} = 0.05$ were well-suited to illustrate the trade-offs between different implementation alternatives. For all our experiments we have further assumed $k_{CM(BM)}$ = 1 as cost factor for the Block RAM.

The experiment shows that the costs for mode adaptivity grows quickly with the number of configurations. On the other hand the increase in costs for just-in-time adaptivity is much less and depends mostly on the cost for the configuration memory. Since the required FIFO depth was much smaller than the minimal size of a Block RAM, an implementation with Distributed RAM has significantly lower costs than an implementation using Block RAM.

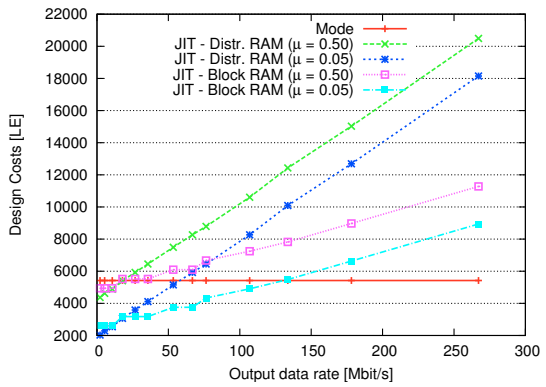

Figure 7. Costs for varying required output data rate

Figure 7 compares just-in-time and mode adaptivity for a varying output data rate requirement for five configurations of the same size. In this experiment, we have used a configuration with a much smaller computation time $t_{C,i} = 60$ ns and a required output data rate of up to $\rho_{out} = 32\text{bit}/(2t_{C,i}) = 267$ Mbit/s for a worst case reconfiguration interval of $t_{interR,i}$ =750μs. The size of the configuration and thus the reconfiguration time is the same as in the first experiment.

The required FIFO buffer space increases with the required throughput. For a throughput of 267 Mbit/s the output FIFO needs to have a depth of 6400 to be able to sustain the required throughput during reconfiguration. The size of the input FIFO buffer does not increase with the required throughput, since it is assumed that the input process delivers at least the same peak input rate as the required output rate.

In contrast to the previous experiment, we observe that the design costs for a high throughput requirement are significantly lower in an implementation with a Block RAM than the corresponding Distributed RAM implementation. This is caused by the large requirement of FIFO buffers, which are not implemented efficiently with Distributed RAMs. There is an almost constant cost for mode adaptivity over the whole throughput range, since it is dominated by constant configuration costs, whereas the costs for the FIFO buffers are minor.

5. Conclusion

We have proposed a novel estimation method for adaptive applications that shall be implemented on a partially reconfigurable FPGA. The main benefit of this method is that it calculates the required buffer sizes for a given required output data rate and enables accurate cost estimates without using back-end tools for partially reconfigurable FPGAs. We have demonstrated our method to experimentally analyse the non-obvious trade-off between just-in-time and mode adaptivity and different buffer memory implementations. Our next step is to extend our method to be able to analyse multiple adaptive processes in a single system, which requires to analyse the contention that will arise at the single reconfiguration port of the FPGA.

References

[1] J. Zhu, I. Sander, and A. Jantsch, "Performance analysis of reconfiguration in adaptive real-time streaming applications," in *6th IEEE Workshop on Embedded Systems for Real-time Multimedia (ESTIMedia)*, October 2008.

[2] *Early Access Partial Reconfiguration User Guide (UG208)*, Xilinx, Inc., 2006. [Online]. Available: http://www.xilinx.com/

[3] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, January 1987.

[4] A. Schallenberg, F. Oppenheimer, and W. Nebel, "OSSS+R: Modelling and Simulating Self-Reconfigurable Systems," in *Field Programmable Logic and Applications*, August 2006.

[5] FOSSY, Website, http://system-synthesis.org.

[6] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Improved functional simulation of dynamically reconfigurable logic," *Lecture Notes in Computer Science*, vol. 2438, 2002.

[7] K. Tiensyria, Y. Qu, Y. Zhang, M. Cupak, L. Rynders, G. Vanmeerbeeck, K. Masselos, K. Potamianos, and M. Pettisalo, "SystemC and OCAPI-xl Based System-Level Design for Reconfigurable Systems-on-Chip," *Forum on Specification and Design Languages*, vol. 2, 2004.

[8] S. Ghiasi and M. Sarrafzadeh, "An Optimal Algorithm for Minimizing Runtime Reconfiguration Delay," *ACM Trans. on Embedded Computing Systems*, vol. 3, May 2004.

[9] Y. Qu, J.-P. Soininen, and J. Nurmi, "Improving the Efficiency of Run Time Reconfigurable Devices by Configuration Locking," *Design, Automation and Test in Europe (DATE'08)*, Mar. 2008.