

# System Integration of Tightly-Coupled Reconfigurable Processor Arrays and Evaluation of Buffer Size Effects on Their Performance

Vahid Lari, Frank Hannig, and Jürgen Teich  
 Hardware/Software Co-Design, Department of Computer Science  
 University of Erlangen-Nuremberg, Germany  
 Email: {vahid.lari, hannig, teich}@cs.fau.de

**Abstract**—This paper studies the loosely integration of application accelerators consisting of an array of tightly-coupled lightweight reconfigurable processors into a system-on-a-chip. In order to explore a multitude of design variations a C++ simulation model of the accelerator has been integrated with a system-on-a-chip environment consisting of a general purpose processor, a DMA controller, an interrupt controller and a memory module. Dependent on the applications, different kinds of I/O buffers are designed around the processor array and the effects of the buffer size on the overall execution time are evaluated. The evaluations are based on new mathematical estimation models derived from the system and application constraints. The estimations are validated with experimental results with an error less than 1%. Exploring several designs points that using our architecture along with suitable buffer sizes, can improve the system execution time, one to two magnitudes for the selected algorithms.

**Keywords:** System-on-a-chip, System performance evaluation, Double buffering mechanism, Virtual system prototyping, Coarse-grained reconfigurable architectures

## I. INTRODUCTION

Steady improvements in the semiconductor industry and the growing request for real-time or near real-time speeds in the application area, signal to have more and more processing elements on a system-on-a-chip (SoC). With the need of executing different applications on such hardwares, new generations of hardware architectures were introduced called reconfigurable architectures.

These architectures allow the customization of reconfigurable processing units in order to meet the specific computational requirements of different applications [1]. Consequently, reconfigurable architectures have nearly the flexibility of general purpose processors combined with the performance, speed, and power consumption close to application specific integrated circuits (ASICs). As a result, integrating a reconfigurable architecture with a general purpose processor may offer better performance and flexibility than a general purpose processor alone.

Considering the functionality of a single processing unit, bit width, and configurability of the interconnections, reconfigurable architectures can be classified in two major categories. The first category are fine-grained reconfigurable architectures that are based on look-up tables (LUTs). Together with an

extremely flexible interconnection network, look-up tables constitute such modern reconfigurable architectures, like field programmable gate arrays (FPGAs). These architectures suffer from a huge reconfiguration data stream [2], inefficient area usage [3] and complex computations for placement and routing algorithms. Examples of reconfigurable systems using FPGAs are cryptographic applications [4], video communications [5], and neural computing [6].

The second category are coarse-grained reconfigurable architectures, based on several functional units (FUs), which are capable of executing word or subword level operations instead of bit-level ones found in common FPGAs. The coarse granularity reduces the delay, area, power consumption, and especially the reconfiguration time compared with FPGAs, but at the expense of flexibility.

In this paper we propose a system integration process of a new class of coarse-grained reconfigurable architectures called weakly programmable processor arrays (WPPA) [7]. A mathematical performance analysis is introduced for studying buffer size effects on the system execution time. In our analysis the influences of the general purpose processor (as the master controller of the system) on the system performance are assessed. The mathematical analysis is evaluated using selected case study applications and their results are compared with the experimental results. Finally, the speedup gained by using our architecture compared to pure software implementation of the applications on the general purpose processor is evaluated.

Section II gives an overview of related works. Section III presents the WPPA architecture and other system components. In Section IV, the system operation is explained. Regarding to the system operation, a mathematical model is presented in Section V to estimate the execution, and the results are compared with experiments in Section VI. Finally, the paper is concluded in Section VII.

## II. RELATED WORKS

With the increasing interest on reconfigurable computing, many coarse-grained architectures have been proposed [2]. Some examples of coarse-grained architectures are Morphosys [1], MATRIX [8], RaPiD [9] and REMARC [10]. Considering the great flexibility features of reconfigurable architectures and their ability of executing different applications,



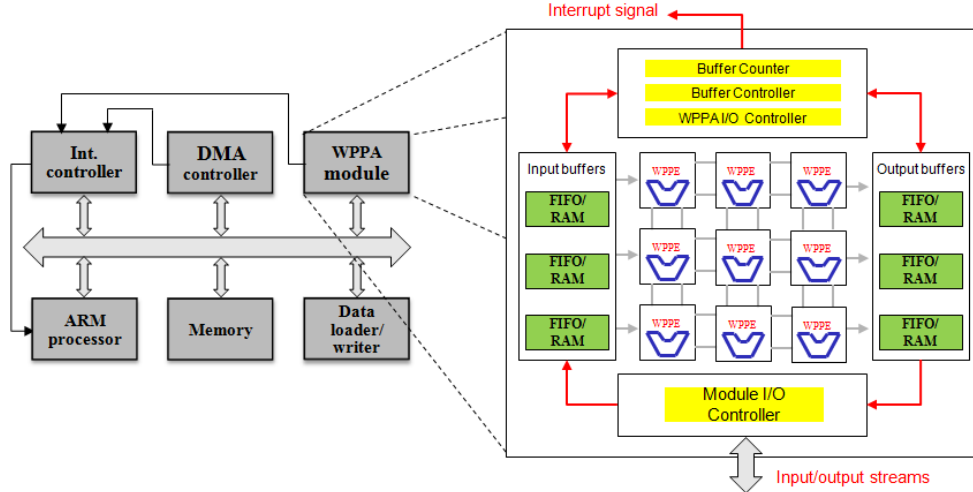


Fig. 2. The system components and the WPPA module structure containing the WPPEs array, I/O buffers and buffer controllers.

nals. After triggering the first input transmission, the system controlling software enters to the wait state where it waits for incoming interrupt signals. After receiving an interrupt, the interrupt handler procedure reacts to it by detecting the interrupt sender (the WPPA module or DMA controller) with the help of the interrupt controller, and initiating a suitable data transmission depending on the WPPA state. The data transmissions between the WPPA module and the memory (or data loader/writer module) are done by the DMA controller. This device has 16 prioritized transfer channels; the channels with lower number have higher priority. Different transfer channels can be dedicated to different devices (or different transmission paths) in the system. Consequently, the transmission initiation delay will be decreased during the system operation.

#### IV. SYSTEM OPERATION

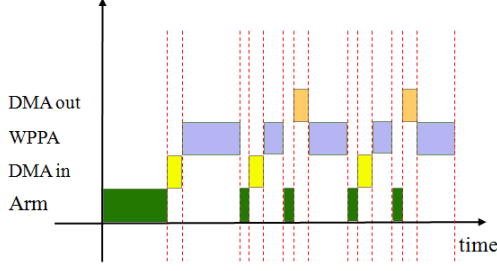
As aforementioned, controlling the system operation is done by an ARM processor. Whereas limited amount of the data buffers can be used inside the WPPA module, the system controlling software should initiate the data transmissions for the input/output buffers in the case of buffer events (input buffer empty or output buffer full).

Inside the WPPA module, as shown in Fig. 2, a buffer controller is implemented that supervises the input/output buffers. This controller requests data transmissions depending on the buffer size and the number of read/write values (that are kept in buffer counters) by sending an interrupt signal to the ARM processor. The input/output buffers are accessed by the system components through the I/O controller of the WPPA module. When a buffer event happens, two strategies can be followed: Halting the computations and waiting for the suitable data transmission or continuing the computations without any halt by using a double buffering mechanism.

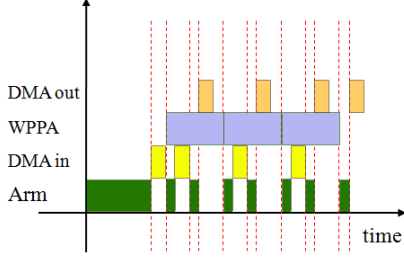
In the first case, after observing an input buffer empty or output buffer full event, the WPPA module halts the computations and sends an interrupt signal to the ARM processor. After a complete I/O data transfer, the WPPA module will automatically resume the computations (see Fig. 3(a)). Using the double buffering mechanism makes it possible to read and write on separate buffer blocks simultaneously. That is, the WPPA can read from an input block while it is pre-fetching another block (or writing on an output block while transferring the last output block) without halting the computations. Toggling between these two buffer blocks is done by the buffer controller with the co-operation of the I/O controller. By using this mechanism, the computation time and the transmission time can be overlapped (see Fig. 3(b)) that can improve the system performance significantly, but—as it will be explained in our mathematical analysis—there are some constraints on using this mechanism.

#### V. SYSTEM EXECUTION TIME

Our main objective in this paper is to find a tradeoff between the buffer size and the system performance. There are some constraints that can affect the system performance; some are forced by system specifications and some are forced by the application nature [11]. The system constraints can be listed in terms of system bus bandwidth, number of concurrent DMA channels and the delay time that elapsed for the interrupt handler procedure in the ARM processor. The input reading and output writing bandwidth of the application are two major factors imposed by the application nature that affect the system performance. We assume that the application is periodic, that means data are fed into the array and generated by it periodically in fixed-sized blocks. This assumption is met by a large range of streaming applications, including encryption [14], compression, and multimedia (image, sound,



(a) Non-overlapping



(b) Overlapping

Fig. 3. System operation

video) processing algorithms [15]. In order to mathematically evaluate the buffer size effects on the execution time, the following notations are used:

- $D_{in}$ : Total input data size
- $D_{out}$ : Total output data size
- $D_{in\_buff}$ : Inputbuffer size inside the WPPA module
- $D_{out\_buff}$ : Output buffer size inside the WPPA module
- $B_{bus}$ : Transmission bandwidth of the bus
- $B_{WPPA\_in}$ : Input data reading bandwidth of the application mapped to WPPA (data samples per cycle)
- $B_{WPPA\_out}$ : Output data writing bandwidth of the application mapped to WPPA (data samples per cycle)
- $T_{ARM\_int}$ : Average time latency elapsed by the interrupt handler procedure in the ARM processor
- $T_{comp}$ : Total computation time elapsed in WPPA in order to do the computations
- $T_{exec}$ : Total execution time of the system

Dependent on the application, the data can be transmitted in 2 modes:

**One-dimensional transmission:** In this mode the data are transmitted in single thread streams. Depending on the data dependencies and the data reusing characteristics of the application, a data overlap factor can be considered. The overlapped data is the amount of data that is retransmitted in two successive data streams. Assuming  $L_D$  be the original data length,  $L_p$  be the partial data transmission length (equal to the buffer size) and the  $L_{ovrlp}$  be the overlapping data length, then the total data transmission length is:

$$L_{total} = L_D + (N_{tr} - 1) \times L_{ovrlp} \quad (1)$$

Where  $N_{tr}$  is the number of data transmissions:

$$N_{tr} = \frac{L_D - L_{ovrlp}}{L_p - L_{ovrlp}} \quad (2)$$

**Two-dimensional transmission:** In this mode the data are partitioned in rectangular tiles, examples of using this transmission scheme are image processing applications and matrix operations. Here also depending on the application, tiles may be overlapped vertically or horizontally, e.g., boarder treatment of adjacent tiles for the edge detection algorithm. Assuming  $H_D$  and  $W_D$  be the height and width of the original data, respectively.  $H_p$  and  $W_p$  are assumed to be the height and width of the tile and  $H_{ovrlp}$  and  $W_{ovrlp}$  are assumed to be the vertically and horizontally overlap factor, respectively. Then, the total height and width of the transmitted tiles will be:

$$H_{total} = H_D + (N_{row\_tiles} - 1) \times H_{ovrlp} \quad (3)$$

$$W_{total} = W_D + (N_{col\_tiles} - 1) \times W_{ovrlp} \quad (4)$$

Where  $N_{row\_tiles}$  is the number of rows of tiles and  $N_{col\_tiles}$  is the number of columns of tiles:

$$N_{row\_tiles} = \frac{H_D - H_{ovrlp}}{H_p - H_{ovrlp}} \quad (5)$$

$$N_{col\_tiles} = \frac{W_D - W_{ovrlp}}{W_p - W_{ovrlp}} \quad (6)$$

And, the total number of tiles is:

$$N_{tr} = N_{row\_tiles} \times N_{col\_tiles} \quad (7)$$

As it has been described, two different strategies are considered for the system operation. The execution time of the system without overlapping is equal to the total time duration spent for input/output data transactions and the computations inside a WPPA. As each data transaction consists of a time duration elapsed by the interrupt handler procedure in the ARM processor plus the data transmission time in DMA, the total execution time of the system is:

$$T_{exec} = N_{in\_tr} \times (T_{ARM\_int} + T_{in\_p}) + T_{comp} + N_{out\_tr} \times (T_{ARM\_int} + T_{out\_p}) \quad (8)$$

Where  $N_{in\_tr}$  and  $N_{out\_tr}$  are the number of input and output data transmissions, which can be calculated by Eq. (2) for one-dimensional transmissions or by Eq. (7) for two-dimensional transmissions. It should be noted that usually for output data transmissions no data overlap factor is considered.  $T_{in\_p}$  and  $T_{out\_p}$  are the DMA transfer time for input or output data and can be calculated using the following equations:

$$T_{in\_p} = \frac{D_{in\_buff}}{B_{bus}} \quad (9)$$

$$T_{out\_p} = \frac{D_{out\_buff}}{B_{bus}} \quad (10)$$

In fact the system operation consists of several partial computation durations. Each of them is equal to the time duration needed to do the computations on one complete input data

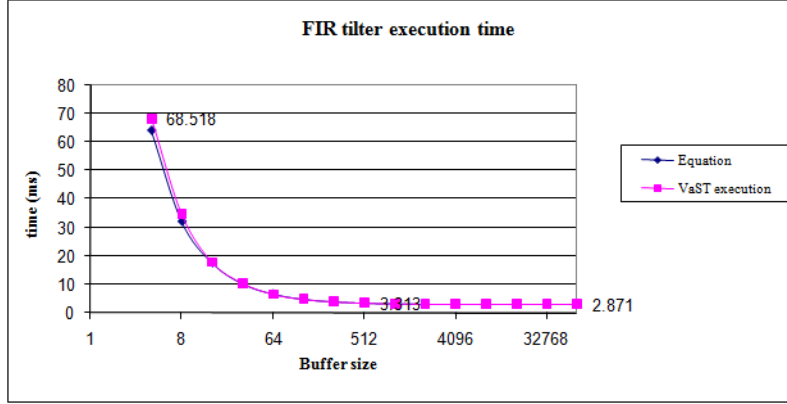


Fig. 4. Diagram of the system execution time for the 6-tap FIR filter

parcel. The number of the partial computations and their duration vary depending on the buffer size, but the total computation time for a certain application and certain amount of input data is fixed, so in Eq. (8) the total computation time is used instead of the partial computation durations. In the non-overlapping mode, the system operates sequentially and for any buffer event the computation is halted until the end of the data transmission. This has a negative effect on the system performance, especially for small buffer sizes that need many data transactions (including interrupt procedure delay and transmission time). Using a secondary buffer can help the system to transfer the data while it is using the other buffer for reading/writing data. The objective of using this scheme is to overlap the computation time with the data transaction time and consequently, to reduce the total execution time. It means that the partial computation time should cover the transaction time for one input parcel and one output parcel. Depending on the number of concurrent transmission channels for input or output transmissions,  $T_{p\_comp}$ , the partial computation time should fulfill the following conditions:

$$T_{p\_comp} \geq (T_{ARM\_int} + T_{in\_p}) + (T_{ARM\_int} + T_{out\_p}) \quad (11)$$

$$T_{p\_comp} \geq T_{ARM\_int} + \max\{T_{in\_p}, T_{out\_p}\} \quad (12)$$

Where the inequality in Eq. (11) is for single channel and the inequality in Eq. (12) is for multiple channel. If the partial computation time fullfills the mentioned conditions, the system execution time will be:

$$T_{exec} = (T_{ARM\_int} + T_{in\_p}) + T_{p\_comp} + (T_{ARM\_int} + T_{out\_p}) \quad (13)$$

The system execution in this equation consists of the system initialization and the first input data transmission, doing the computations inside WPPA parallelly with data transmissions and finally the last output data transmission.

## VI. EXPERIMENTAL RESULTS

In order to evaluate the system, a platform is setup using the devices that have been explained in Section 3. The frequency of the system core clock and also the bus clock are both 100MHz ( $T_{clk} = 10nsec$ ). The devices are connected together

using a standard bus that transfers 4 bytes every 4 cycles. A single transmission channel for both input and output transmissions is prepared for the WPPA module.

As the case studies, two applications are implemented on the WPPA architecture: A 6-tap FIR filter and an edge detection algorithm. These applications have been selected as our case studies because they need different kind of input/output buffers and different transmission modes. The FIR filter is implemented on 6 WPPEs that are connected together in a single pipelined manner. The single thread input data are fed into the pipeline, one input data per cycle. Also after passing the pipeline length, single output will be generated, one output data per cycle. As input/output buffers, single thread FIFOs are used for input and output data streams; consequently one-dimensional data transmission is used as data transmission scheme. In order to evaluate the effects of the FIFO size on the system performance, the application has been executed with different FIFO sizes. The results of the system execution time are shown in Fig. 4.

Due to the growing increase of the number of data transactions for the small buffer sizes, the execution time for the small buffer sizes increases exponentially. This is because of the interrupt delay time that is needed in each data transaction; as the number of data transactions becomes more, the total delay time elapsed by the interrupt procedure becomes longer. Consequently, for buffer sizes greater than 256 bytes that need less transactions, the execution time does not change significantly.

In the same diagram the system execution time estimation for different buffer sizes is also shown (using Eq. (8)). The estimated time for small buffer sizes, especially for 4 or 8 bytes, differs from the experimental results. This is because of that we have used an average time for the interrupt delay; consequently for small buffer sizes that need many data transactions, any small difference between the used value with the actual value can affect the final result significantly. The average estimation error for the presented equation is about 0.93%. The 6-tap FIR filter has been implemented also on the ARM processor purely in software. The system execution time

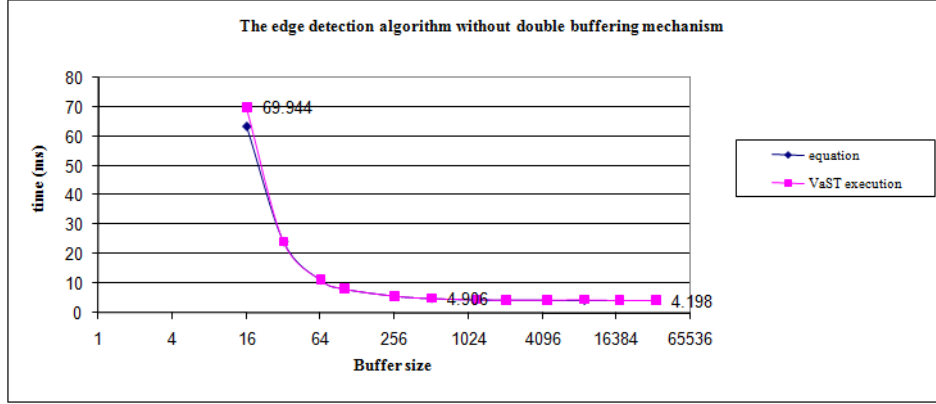


Fig. 5. Diagram of the system execution time for the edge detection algorithm without using double buffering mechanism

using this implementation is about 35 milliseconds. Using the hardware approach with buffer sizes greater than 256 bytes, we can execute the system more than 10 times faster than the pure software implementation. On the other hand, for buffer sizes smaller than 8 bytes, using the hardware approach is not worthy. For the FIR filter, the application and system constraints do not fulfill the inequality in Eq. (11), because the partial computation time for any buffer size is less than the summation of one input and one output data transaction time.

As the second case study, an edge detection algorithm (Sobel operator) has been implemented on the WPPA architecture. The algorithm is computed over 3\*3 pixel windows, consequently for any output, 9 inputs should be read. This window slides over the input picture to generate all the output pixels. Every 3 cycles one input window is read (3 pixels per cycle) and also every 3 cycles one output pixel is generated. For this application, RAM buffers and two-dimensional transmissions are used because of data reusing nature of the application. As the adjacent tiles have shared data in their borders, for this application the vertical and horizontal data overlap factors should be considered. The execution times for this application are shown in Fig. 5 for different buffer sizes. The sizes of the tiles are chosen in a manner that the picture is partitioned into the complete tiles. As it is shown in the diagram for buffers bigger than 1024 bytes, the execution time does not change significantly. In this diagram the estimation of the execution time (calculated by Eq. (8)) is also presented. Like for the FIR filter, there is a gap between the estimated time and the experimented time due to the use of an average delay value for the interrupt handler. Here, the average error of the estimation is about 1.29%.

To investigate the possibility of using the double buffering mechanism, the inequality in Eq. (11) for a given and should be solved. Whereas, the partial computation time for a given buffer size is equal to:

$$T_{p\_comp} = (W_p - W_{ovrlp}) \times (H_p - H_{ovrlp}) \times T_{clk} \times 3 \quad (14)$$

So, the inequality in Eq. (11) will turn to:

$$(W_p - W_{ovrlp}) \times (H_p - H_{ovrlp}) \times T_{clk} \times 3 \geq 2 \times T_{ARM\_int} + W_p \times H_p \times T_{clk} + (W_p - W_{ovrlp}) \times (H_p - H_{ovrlp}) \times T_{clk} \quad (15)$$

Fig. 6 shows the execution times after applying this mechanism, along with the mathematical estimations. In this diagram the total size of double buffers is considered and the results are compared with corresponding buffer sizes without using double buffering. Using this mechanism improves the execution time of the edge detection algorithm in average about 45%.

Despite the results of the single buffer, there is a slight increase for execution time respect to the buffer size. This contradiction can be explained using Eq. (13) where the execution time consists of three parts: First input tile transmission, computation time, and the last output tile transmission. The total computation time for a given application and amount of input data is fixed, so changing the tile size has no effect on it. But the first input and the last output tile transmission times increase with the tile size. Consequently, the total execution time will be increased. It shows the importance of inequalities that mentioned in Eq. (11) and Eq. (12), which can be used to calculate the smallest buffer size that suits for the double buffering mechanism. As an example, as shown in Fig. 6, for a buffer size equal to 1024 bytes, the best execution time is gained. In addition, the edge detection algorithm is implemented by software on the ARM processor. Here, the total system execution time for this implementation is 412 milliseconds. The hardware approach even with smallest buffer size computes the application about 6 times faster than the software approach. The speedup for a buffer size equal to 100 bytes is about 50 times, for a buffer size equal to 512 bytes is 84, and for a buffer size of 1024 bytes is 92. Using double buffering with a total buffer size of 1024 bytes, the hardware approach can compute the application 145 times faster than the software approach.

## VII. CONCLUSION

In this paper the system integration of weakly programmable processor arrays has been studied. The considered

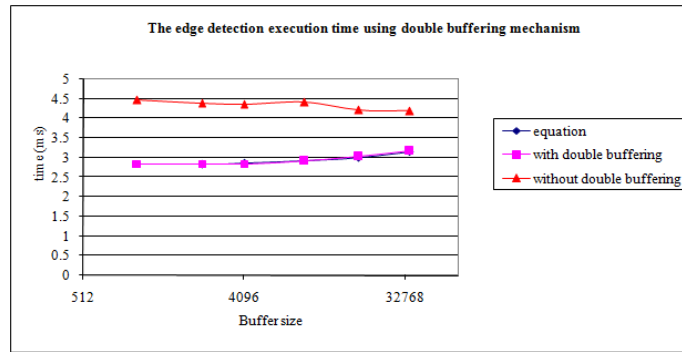


Fig. 6. Diagram of the system execution time for the edge detection algorithm using double buffering mechanism

system consists of an ARM processor, a DMA controller, an interrupt controller, memory modules, a standard bus, a WPPA module, and a test bench image loader/writer module. As I/O buffers, depending on applications, FIFO buffers or RAM buffers were implemented. In order to evaluate the buffer size effects on the system performance, a mathematical model, was introduced for system execution time estimation. As case studies, two applications were implemented: A 6-tap FIR filter and an edge detection algorithm. For the FIR filter, changing the buffer size had a great influence on the execution time for buffer sizes less than 256 bytes, while it had not a significant effect for buffer sizes greater than 256 bytes. The mathematical model could estimate the execution time of this application with an error less than 0.93%. Like the FIR filter, the system execution for the edge detection algorithm reached a steady value for the buffer sizes bigger than 1024 bytes. The estimation error using the mathematical model was 1.29%. The system was also evaluated using the double buffering mechanism, which improved the execution time in average 45%. Using our mathematical analysis along with a little knowledge about the timing delay imposed by the general purpose processor (elapsed by interrupt the handler procedure), helps us to explore numerous system designs in a really short time. In addition, for double buffering mechanism, using our methods help to find the smallest buffer size with the best execution time. Our future work focuses on extending the double buffering idea over bigger range of buffer sizes, executing multi-applications on a single array and arranging efficient buffering systems for different applications on an array.

#### VIII. ACKNOWLEDGEMENT

This work has been supported in part by the German Science Foundation (DFG) in project under contract TE 163/13-2.

#### REFERENCES

- [1] H. Singh, M. Lee, G. Lu, F. Kurdahi, N. Bagherzadeh, and E. Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.
- [2] R. Hartenstein, "A Decade of Reconfigurable Computing: A Visionary Retrospective," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. Munich, Germany: IEEE Computer Society, Mar. 2001, pp. 642–649.
- [3] A. Singh and M. Marek-Sadowska, "FPGA interconnect planning," in *Proceedings of the international workshop on System-level interconnect prediction (SLIP)*, California, USA, Apr. 2002, pp. 23–30.
- [4] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable active memories: Reconfigurable systems come of age," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 1, pp. 56–69, Mar. 1996.
- [5] J. Villasenor, C. Jones, and B. Schoner, "Video communications using rapidly reconfigurable hardware," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 6, pp. 565–567, Dec. 1995.
- [6] J. Eldredge and B. Hutchings, "Density enhancement of a neural network using FPGAs and run-time reconfiguration," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, California, USA, Apr. 1994, pp. 180–188.
- [7] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich, "A Highly Parameterizable Parallel Processor Array Architecture," in *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. Bangkok, Thailand: IEEE, Dec. 2006, pp. 105–112.
- [8] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, California, USA, Apr. 1996, pp. 157–166.
- [9] C. Ebeling, D. Cronquist, and P. Franklin, "Configurable computing: the catalyst for high-performance architectures," in *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, Seattle, USA, Jul. 1997, pp. 364–372.
- [10] T. Miyamori and K. Olukotun, "A quantitative analysis of reconfigurable coprocessors for multimedia applications," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, California, USA, Apr. 1998, pp. 2–11.
- [11] E. El-Araby, M. Taher, K. Gaj, T. El-Ghazawi, D. Caliga, and N. Alexandridis, "System-level parallelism and throughput optimization in designing reconfigurable computing applications," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS) - Workshop 3*, Santa Fe, New Mexico, USA, Apr. 2004, pp. 26 – 30.
- [12] A. Kupriyanov, D. Kissler, F. Hannig, and J. Teich, "Efficient Event-driven Simulation of Parallel Processor Architectures," in *Proceedings of the 10th International Workshop on Software and Compilers for Embedded Systems (SCOPE)*. Nice, France: ACM Press, 2007, pp. 71–80.
- [13] VaST Systems Technology Corporation, <http://www.vastsystems.com>, March 2009.
- [14] O. Fidanci, D. Poznanovic, K. Gaj, T. El-Ghazawi, and N. Alexandridis, "Performance and overhead in a hybrid reconfigurable computer," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, Apr. 2003, pp. 22 – 26.
- [15] K. Parhi, *VLSI digital signal processing systems*. Wiley New York, 1999.