

Evaluation of Computing in Memory Architectures for Digital Image Processing Applications

David Landis, Paul Hulina, and Scott Deno
The Pennsylvania State University
Center for Design & Computing
11 EE West, University Park, PA 16802
dll2@psu.edu

Luke Roth and Lee Coraor
The Pennsylvania State University
Dept. of Computer Science & Engineering
220 Pond Lab, University Park, PA 16802
roth@cse.psu.edu

Abstract

Continuing improvements in semiconductor density are enabling new classes of System-on-a-Chip architectures that combine extensive processing logic and high-density memory. Many of the capabilities of these architectures can be custom tailored to the demands of real-time image processing. This paper identifies and describes candidate computing in memory architectures, and evaluates their performance on several image-processing algorithms.

1. Computing In Memory Technology

By 2009, the Semiconductor Industry Association's Technology Roadmap predicts that a single high-end microprocessor die will contain approximately 84 million logic transistors [1]. Dynamic Random Access Memory (DRAM) density is increasing at an even more rapid pace. By 2009, a state-of-the-art DRAM chip is expected to have a capacity of 2Gbytes. Larger scale problems will be easily handled "on-chip" using these tremendous increases in memory and logic density.

The concept of integrating computing logic and high-density DRAM on a single die has been given many names: Processor in Memory (PIM [2]), Intelligent RAM (IRAM [3]), Computational RAM (C-RAM [4]), Parallel Processing RAM (PPRAM [5]). We refer collectively to these new ways of integrating memory and logic as Computing In Memory Architectures (CIMA). This paper examines CIMA alternatives, and considers their applicability to digital image processing problems.

1.1 Characteristics of CIMA Designs

The integration of DRAM and computing logic on the same Integrated Circuit (IC) die gives CIMA designs several desirable characteristics. First, the physical size and weight of the overall design can be reduced. As more functions are integrated on each chip, fewer chips are needed for a complete design. Second, because the DRAM is located on the same die as the computing logic,

very wide on-chip buses between CPU and memory can be used. Third, the elimination of off-chip drivers has the effect of reducing power consumption and latency. In spite of hybrid process limitations, Siemens corporation [6] states that their new embedded DRAM process will achieve a cycle time as low as 7ns, with a bus width of up to 128 bits per page. These combine to produce an on-chip single-bank bandwidth of nearly 2.3 Gbytes/second.

The integration of DRAM and logic on the same die presents several challenges. First, high-volume semiconductor processes are highly optimized for the particular characteristics and features demanded by the product, e.g. high switching rates for logic processes, and long retention times for DRAM processes. A hybrid manufacturing process must compromise between these two characteristics, leading to sub-optimal performance – estimates have ranged from a 30% to 100% decrease in overall logic speed [3]. However two major manufacturers (Siemens and IBM) have recently announced first-generation mixed processes that claim little to no performance reduction [7, 8].

While overall system power consumption will decrease due to the elimination of external driver circuitry and buses, total on-chip power consumption may actually increase. This presents a problem for integrated DRAM, since higher junction temperatures reduce DRAM retention times. In addition, DRAM manufacturers rely on external testing to find and route around defective memory cells. This built-in redundancy allows DRAM manufacturers to achieve greater yields and lower per-unit costs than a microprocessor of equivalent die size. A CIMA with a large amount of DRAM must therefore provide either on-chip testing circuitry, or special support to give external test equipment access to DRAM circuitry.

1.2 Why Digital Image Processing?

CIMA designs have been proposed for use in many fields, from handheld to high-performance parallel computing systems. Few applications, however, are quite as well suited to the unique strengths and weaknesses of

CIMA as digital image processing. Image processing applications generally require high-bandwidth, low-latency access to image data. The algorithms have significant parallelism, and require a large number of very simple computations.

A CIMA design can provide the bandwidth and latency needed by image processing applications without resorting to interleaving of multiple banks of DRAM ICs. However, the reduced logic switching speeds available in a hybrid (logic/DRAM) fabrication process would be more suitable to a design containing multiple simple execution units than to a single high-speed CPU. This type of design would be expected to perform much better on the highly parallel algorithms of image processing than on general-purpose desktop or scientific applications.

2. Digital Image Processing Operations

The term “Digital Image Processing” (DIP) refers to a wide variety of applications, from photo restoration to video compression. Many applications require that this processing be completed in real time, as when the image data is being generated by a camera or other sensor and must be displayed to the user. This leaves a fixed amount of time – typically 1/30 of a second – for the complete processing of an image containing over a million pixels.

2.1 Image Processing Primitives

Most DIP algorithms can be decomposed into simpler operations, which are commonly categorized as global, local, or point operations. Global image processing operations produce an output in which every value is a function of every pixel in the original image. An example is a histogram operation, in which the number of times each individual intensity value or range of intensity values occurs in the image is counted. This class of operations tends to be the most difficult to parallelize.

A local, or windowed, operation produces an output that depends on the pixels in the local area about a pixel. An example of such an operation is an averaging filter, which can be used to smooth out grainy images. While some local operations are more easily parallelizable than others, it is generally possible to process multiple windows in parallel, speeding up the computation. Finally, point (or *pixel*) operations produce an output (generally a second image) in which each output value is a function of only one pixel of the image. These operations are the most easily parallelizable – in the extreme, every pixel could be processed simultaneously.

2.2 Common Image Processing Features

Although image-processing operations may access memory in different patterns, they share some very

important features. First, they are all very memory-intensive. Traditional computing applications perform a significant amount of processing on each piece of data before loading the next. However, in image processing at least one new pixel of information is typically needed for each step in the computation. According to one study using traditional DSP’s and a variety of common algorithms (threshold detection, erode/dilate, median, convolve, and others) there are up to two memory accesses for each floating-point instruction [9].

Next, the temporal locality of the image-processing operations is minimal. When a pixel is accessed for a local or global operation, it might never be used again until the operation is repeated. In windowed operations, the pixel is only needed for the duration of the time it is in the window; when the window moves past a pixel it will not be accessed again until the window reaches the end of the current row. Thus, traditional caching schemes cannot effectively speed up or reduce the number of memory accesses. However, the spatial locality of image processing operations tends to be quite high; if a certain pixel of a frame buffer is accessed, the most likely next accesses are to its immediate neighbors. Because of the two-dimensional nature of the image (plus the added dimension of time in video processing), pixels that are adjacent in the image may not be located at adjacent memory addresses. This reduces the locality of the data unless complicated memory addressing is used.

Finally, the individual computations used in each operation are often quite simple, frequently involving only 8-bit data. Even 24-bit color data is comprised of 8-bit channels. This implies that using a 32- or even 64-bit processor to do image processing tasks can be very inefficient unless the processor contains specialized instructions for dealing with this type of data. Worse yet, many common operations, such as erosion and dilation filters, typically operate on simple two-level binary data.

2.3 CIMA Application to Image Processing

The most immediate benefit that CIMA offers for image processing applications is the increased on-chip memory bandwidth and lower memory latency. Consider an on-chip DRAM that can transfer 128 bits every 6 nanoseconds (performance comparable to that claimed by current first-generation mixed fabrication processes). In this scenario, an entire 1-MB frame buffer could be streamed through memory in less than 1/2000 of a second – a pixel-processing rate of 2.67 billion 8-bit pixels/second. This, of course, requires a processing unit fast enough to keep up with this tremendous bandwidth. Various techniques have been proposed to take advantage of this bandwidth, from on-chip parallel processing (PPRAM at Kyushu University, Japan [5]) to a modified vector processor (the IRAM project at U.C. Berkeley,

USA [3]). It is clear is that any proposed CIMA design will have to provide a great deal of computational parallelism to exploit the full bandwidth available. At a conservative clock rate of 166MHz, at least 16 pixels (at 8-bit depth) are available to be processed each cycle.

The lack of temporal locality in most image processing applications makes traditional data caching systems ineffective. Because of this, a CIMA design optimized for image processing applications would likely forego an extensive cache, perhaps using only a small instruction cache and no data cache. While this can adversely affect performance on those few image-processing algorithms that do show temporal locality, this would be largely compensated for by the low latency of the DRAM. The large bus widths available from an on-chip DRAM are well matched to the spatial locality of an image frame buffer. With a bus width of 128 bits, a window of 16 pixels could be loaded in a single DRAM cycle.

Due to the lower on-chip logic density of a mixed DRAM/logic process, it is impossible for a CIMA design to incorporate all of the functionality of a state-of-the-art microprocessor together with enough DRAM to provide extensive program and image data storage. Fortunately, because of the relatively coarse quantization associated with most image processing operations (8-16 bit), the computational power of a modern 32- or 64-bit processor with floating-point math capabilities is seldom needed. Instead, an array or vector of simpler 8-bit Processing Elements (PEs) tailored to the specific needs of image processing applications is more efficient.

3. CIMA Architectural Options

Thus far we have considered generalized CIMA designs as a group. However, the range of designs that fall under this architectural style is quite rich and varied. Over the past few years, many architectures have been proposed; each embodies a distinct subset of the CIMA advantages and disadvantages. Of these, four major design styles represent the majority of the variations on the CIMA theme: the Processor-In-Memory (PIM), the Vector DRAM Processor, the Multiprocessor-on-a-chip, and the merged CPU/FPGA/DRAM. The four block diagrams contained in Figure 1 illustrate the basic internal organization of each of these CIMA architectural styles.

3.1 Processor-in-Memory

One of the first of the new generation of CIMA designs to be proposed and built was the C-RAM design [4], developed by the University of Toronto in 1992. A similar design, the Terasys PIM array [2], was announced in 1995. This design style features a large array of simple computation elements (typically over 1,000) which are built into the DRAM arrays. These processing elements

are usually integrated at the output of the sense amplifiers and are controlled by a single control unit, as an SIMD processor. This style of CIMA design is generally the most capable of exploiting the enormous on-chip bandwidth of the DRAM, as the computation elements are integrated directly into the DRAM outputs. From an architectural standpoint, this design is also the simplest and can achieve the highest theoretical performance.

However, the Processor in Memory approach also has some serious drawbacks. First, though architecturally simple, serious complications arise in the actual design and fabrication. Most DRAM cores are highly optimized, and can only be modified with difficulty. While prototype systems developed using in-house designs may be easily modified, it is difficult to achieve the same level of integration with the highly optimized designs used by the large DRAM manufacturers. Second, these types of massively parallel, simple-processing-element SIMD designs have so far only been successful on a limited set of applications that can be simply parallelized. For applications with a significant amount of serial computation, the speedup from this architecture is very limited. Third, because of the tight integration with the DRAM cell, there is space for only limited capability processing elements. The C-RAM and PIM designs both contain only single-bit processors, significantly limiting the set of applications that the architectures can accelerate.

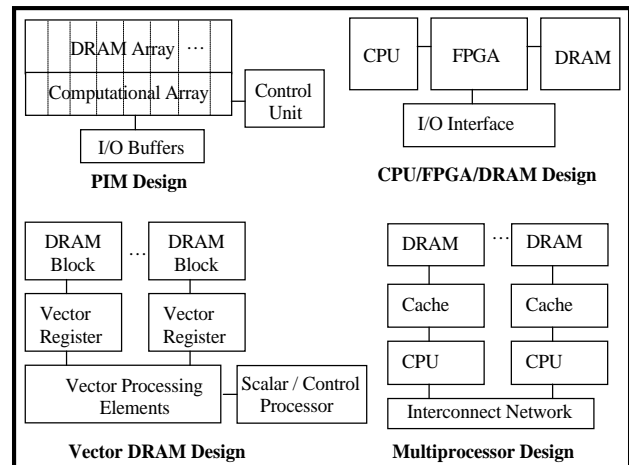


Figure 1. Representative CIMA Architectures

3.2 Vector DRAM

The CIMA design that has received the most published attention to date is the IRAM project at U.C. Berkeley [3]. This ambitious design attempts to take advantage of the high bandwidth and low latency offered by on-chip DRAM by integrating a complete vector processor on a single die. Moving the computation farther from the outputs of the DRAM reduces the peak throughput

capability of the CIMA system. However, by separating the components, fewer but more powerful processing elements can be used. This gives performance improvements on a larger set of applications, and allows a better-understood programming model to be employed. The IRAM project's stated objective is to build a full Cray T90-class 64-bit vector processor into its design. However, a more modest 8/16 bit vector design would be better suited to the image processing applications considered in this paper.

The IRAM architecture has a much lower peak performance than the PIM architecture due to its smaller number of parallel functional units. However, IRAM still shows a significant speedup compared to a traditional RISC processor implementation. The deep pipelines of traditional vector processors, used to hide the long latency associated with floating-point computations, are not necessary when dealing with 8- and 16-bit integer image data. By simultaneously processing multiple vectors of data and removing a great deal of loop overhead, it is still possible to see a large speedup on many applications.

3.3 Multiprocessor-on-a-Chip

There are several CIMA architectures that fall between the extremes of large numbers of single-bit processors vs. a single highly integrated high performance computer-on-a-chip. Integration of multiple, somewhat simpler, processors onto the same IC offers a number of potential advantages. Such a "single-chip multiprocessor" would benefit not only from increased integration, but also from communication latency and bandwidth that is impossible to attain once signals have moved off the chip. An example of this type of architecture is the PPRAM [5].

The single-chip multiprocessor PPRAM design tries to avoid the central control unit bottleneck that has hampered the usability of other designs. It does so by integrating several relatively simple, but fully independent, cached RISC processors, each with a reasonable amount of memory (8+MB). These processors are connected by a very high-bandwidth interface, and can be programmed using standard shared-memory or message-passing parallel algorithms. Because of the high-level programmability of these designs, they are more easily programmed for maximum parallelism than the other CIMA designs. However, the large amount of resources necessary for each node limits the total number of parallel nodes to two or four per chip in first-generation designs, far short of the 1,000+ parallel processing elements of the previously described PIM design.

3.4 Merged CPU/FPGA/DRAM

A recent contender in the field of high performance computing is the reconfigurable Field Programmable Gate

Array (FPGA). FPGA density has now reached the point where up to 1,000,000 gates can be emulated by a single FPGA at clock rates not far behind that of custom logic. Recently, several designs have been proposed merging CPU and DRAM technology with FPGA logic to create reconfigurable processing units that can adapt to changing processing needs. There are many variations to this design style, from the DRAM/FPGA chip proposed by NEC Corporation [10] (capable of dynamically reconfiguring itself in under 100ns) to the Dynamic Instruction Set Computer (DISC) being developed at Brigham Young University [11].

Generic FPGA-only designs have been used in many digital signal processing and image processing applications. Their reconfigurability allows them to provide precisely those resources needed for the current application. Instead of trying to provide a physical hardware accelerator for every possible application, a software library of accelerators could be provided and loaded into the FPGA as necessary for each individual application. However, the combination of FPGA resources with a traditional CPU provides a guarantee of reasonable performance on algorithms that are not suited for FPGA processing, as well as a simpler means of programming and coordinating the system.

The addition of DRAM to the FPGA design provides a large, fast memory space sufficient to hold many FPGA configurations. This offers fast reprogrammability as well as the capability to locally store reasonable amounts of image data. Using today's fabrication technology, combining a large DRAM, a CPU core, and a sufficiently large FPGA array would result in an unreasonably large die (400-500+mm²). However, with the density available from a next-generation 0.18 micron process, the required die area becomes more reasonable.

4. CIMA Performance Comparison

High-level conceptual models of each of the CIMA architectures were developed in order to compare relative performance in image processing applications. The steps required for the execution of various image-processing algorithms were determined, and the processing time needed for each step in each high-level model was estimated. The models do not take into account precise details of the cycle-by-cycle performance of each design. However, they incorporate a sufficiently detailed behavioral description to provide consistent estimates of relative performance. A variety of algorithms were "simulated" in this fashion; the three reported in this paper are representative of the larger class of image processing applications: histogram (global), 3x3 averaging filter (local), and thresholding (point).

The overall results of this comparison are illustrated in Table 1. Note that idealized execution assumptions were

used for all of these performance estimates. The actual performance would likely be significantly lower for all machines after actual machine limitations (such as cache misses, interconnect delays, synchronization delays, and read/write contention) are taken into account. The timing and die size figures were computed assuming current-generation 0.25 micron fabrication technology, with an image size of 1024x1024 8-bit pixels.

Arch. Type	Histogram	Filter	Thresh hold	Die Area (0.25 u)	Die Area (0.18 u)
RISC CPU	19ms	67ms	17ms	300mm ²	180mm ²
PIM	71ms	0.6ms	0.1ms	350mm ²	140mm ²
Vector DRAM-333MHz	28ms	50ms	0.9ms	430mm ²	190mm ²
Vector DRAM-250MHz	38ms	67ms	1.2ms	430mm ²	190mm ²
Multi-CPU 4 @ 333	8ms	28ms	6.5ms	450mm ²	200mm ²
Multi-CPU 4 @ 250	10ms	37ms	9ms	450mm ²	200mm ²
CFD 8000 LE	73ms	10ms	1.0ms	410mm ²	180mm ²
CFD 16000 LE	37ms	6ms	1.0ms	500mm ²	230mm ²
CFD 8000 LE, fast-carry	23ms	1.5ms	1.0ms	430mm ²	190mm ²
CFD 16000 LE, fast-carry	12ms	1.2ms	1.0ms	530mm ²	250mm ²

Table 1. Performance of CIMA Alternatives

4.1 Performance Estimation - Detailed Example

This section describes how our performance estimates were generated, by showing how the averaging filter performance was estimated. First, a trace of the assembly language program necessary for the inner loop of the filter algorithm was generated using the GCC compiler. This trace contains a total of 36 instructions: 5 memory access, 15 add/subtract, 11 register move, 3 multiply, 1 divide, and a conditional branch. We assume that the processor is sufficiently pipelined and superscalar to average one instruction per cycle, so 36 cycles are needed for each pixel. Efficient execution on a 4-chip system – the largest number of processors that fit in a die area comparable to the other designs – would distribute one quarter of the image (262144 pixels) to each processor for a total of 9.43 million cycles. Because the individual processors are simpler than today’s state-of-the-art, we assume a performance reduction of 30 to 50 percent from our baseline 500MHz RISC, resulting in a 333 - 250MHz clock rate. At that speed, the averaging filter execution time would be 28 - 37ms. This estimate ignores the effects of initialization and startup time; however, with larger images most of the time is spent in this inner loop.

The same assembly trace was used to generate the performance estimate for a single RISC processor, the only differences being the increased computation per processor and the higher 500 MHz clock speed. For the

remaining architectures, however, the simulation was not as straightforward. Without a vectorizing compiler at our disposal, it was necessary to generate the assembly trace for the vector DRAM machine by hand. The increased parallelism available in the vector machine reduced the number of processing cycles per pixel to 19, a clear reduction from the scalar processor.

The PIM processor was modeled as a SIMD array of 1024 single-bit processors, with nearest-neighbor connections and direct single-cycle access to main memory. Because of the simplicity of the individual processors, a total of 10 cycles is needed to perform an 8-bit addition; the eight additions and division by 9 consume a total of 90 processing cycles. The time to process the entire image, then, would be (90 cycles/pixel) * (6 ns/cycle) * (1M pixels) / (1024 processors), or approximately 0.6 milliseconds.

Finally, the CPU/FPGA/DRAM (CFD) design analysis was based upon the performance of contemporary SRAM-based FPGAs. The averaging filter required a minimum of 96 3-input function blocks and could produce one output every 40 cycles. Assuming a time and area penalty of 50% for routing and data alignment, a CIMA design with 8000 function blocks would be able to accommodate 40 of these filters, improving the overall processing rate to 2 pixels/cycle. At 200MHz, our 1MB image could be processed in 10ms.

4.2 Performance Analysis Results

Overall, our results show that each of the CIMA designs has a significant advantage over a RISC processor in at least one of the algorithms. The PIM design showed the most dramatic performance gain, with an averaging filter improvement of two orders of magnitude. However, because its clock rate is locked to the DRAM cycle time, its histogram performance was the lowest of all designs.

For the histogram (or any other algorithm in which there is an insufficient amount of low-level parallelism), execution will need to be carried out by the host processor. While the processor-in-memory design offers immense theoretical performance gains, it suffers from a lack of non-trivial applications on which its full performance can be realized. However, it may be possible to reduce this limitation by adding more capable interconnect between processing units and increasing the functionality of the individual processing units.

The vector DRAM architecture provided a performance improvement on a wider range of algorithms than the PIM design. However, the peak performance improvement over a traditional RISC architecture was not nearly as great as that of most of the other architectures.

The single-chip multiprocessor design showed a balanced speedup on both algorithms, despite the fact that its peak performance was less than either the PIM or the

vector DRAM. Because each processor is capable of independent program execution, this is the easiest design to efficiently program for parallel execution.

The CPU/FPGA/DRAM design appears to offer the most promise for future performance gains. It has the second-highest performance on the highly parallel averaging, however, it is still able to outperform the baseline RISC processor on the histogram algorithm. The performance impact of several FPGA design options for the individual logic blocks can also be seen in the results of Table 1. The addition of fast-carry circuitry, which has become common in recent FPGA designs, significantly increases performance. The CPU/FPGA/DRAM design also contains a traditional integer datapath, which serves to accelerate algorithms containing arithmetic computations such as multiplies and divides; operations that are slow and resource intensive when implemented on an FPGA. This performance, however, comes at a price. Integrating sufficient FPGA logic elements into a CIMA design to improve performance (at least 16MB of DRAM and 8000 LUT's) using current-generation 0.25 micron fabrication technology would require a prohibitively large die (430mm²). However, considering the timeframe for introduction of a design based on this concept, the improved manufacturing technology (0.15 micron in 2001 [1]) would accommodate this architecture in a more reasonably sized (220mm²) die. The flexibility provided by the FPGA also makes this design difficult to program, as each program must specify a hardware configuration as well as software instructions. However, recent research has demonstrated the feasibility of compilers that can generate hardware/software instruction sequences from standard ANSI C code [13].

5. Conclusions and Future Work

The parallel nature of most image processing applications allows each of the CIMA architectures evaluated in this paper to show a distinct performance advantage compared to a RISC architecture fabricated using a similar process technology. However, each design also has its own specific shortcomings that need to be addressed in order to gain widespread acceptance. Of the four architectures, the Processing-In-Memory design, while theoretically the most powerful, has the most obstacles to overcome before it is suitable for all but a few applications. The Vector DRAM design may provide the greatest initial performance benefit, as it balances increased performance with a mature programming model. The Multiprocessor-on-a-chip may perform well in traditional server or workstation systems. However, for image processing applications this architecture does not provide sufficient parallelism to produce a significant speedup. The CPU/FPGA/DRAM design, with borderline practicality under current

fabrication technology, provides the largest potential for future performance improvement with process technology advances.

While our performance estimates indicate a significant advantage for the CPU/FPGA/DRAM design, it is clear that modifications to the basic FPGA logic block have dramatic effects on the performance of the design. As seen in Table 1, the addition of fast-carry logic results in performance over 6 times faster on certain algorithms. Other elements of the design, such as reconfiguration time, bandwidth between the FPGA grid and the on-chip DRAM, and the amount of communication overhead between the FPGA grid and the controlling CPU, are likely to have similarly large effects. Our next step is to focus on this architecture in more detail and determine what combination of features will provide the best possible image processing performance over a wide range of applications. Ultimately, we are working towards a prototype design based on the results of this analysis that will provide balanced, real-time performance over a wide range of image processing applications.

References

- [1] The National Technology Roadmap for Semiconductors, *SIA Semiconductor Industry Association*, 1997 Edition
- [2] M. Gokhale, B. Holmes, K. Iobst. "Processing in Memory: The Terasys Massively Parallel PIM Array", *IEEE Computer*, April 1995 pp.23-31
- [3] D. Patterson, T. Anderson, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, K. Yelick, "A Case for Intelligent RAM: IRAM", *IEEE Micro*, April 1997 pp.34-44
- [4] D. Elliot, W. Snelgrove, M. Sturm, "Computational RAM: A Memory-SIMD Hybrid and its Application to DSP", 1992 *Custom Integrated Circuits Conference*, May 1992 pp.30.6.1-30.6.4
- [5] K. Murakami, K. Inoue, H. Miyajima, "Parallel Processing RAM", Nov.97, <http://kasuga.csce.kyushu-u.ac.jp/~ppram>
- [6] "Embedded DRAM: Innovations that fit", Siemens Semi., 2nded. '97, <http://www.siemens.de/semiconductor/edram>
- [7] IBM Press Release: "IBM chip advance spurs 'system-on-a-chip' products", Feb. 22, 1999
- [8] K. Schoneman, "A Modular Embedded DRAM Core Concept in 0.24 micron Technology", *Proc. 1998 Memory Technology Design and Test Conf.*, 24-25 August 1998
- [9] "Local Memory Enhances Performance of Machine Vision Application", <http://www.alacron.com/news/art04.html>
- [10] M. Motomura et al., "An Embedded DRAM-FPGA Chip with Instantaneous Logic Reconfiguration", 1997 *Symposium on VLSI Circuits*, June 12-14 1997 pp.55-56
- [11] M. Wirthlin, B. Hutchings, "A Dynamic Instruction Set Computer", *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, April 1995 pp. 99-107
- [12] Xilinx 1999 Databook p.161 ff.
- [13] D. Clark, B. Hutchings, "Supporting FPGA Microprocessors through Retargetable Software Tools", *Proc. 1996 Symp. on FPGAs for Custom Computing Machines*, April 1995 pp.195-203