An Evaluation of the Suitability of FPGAs for Embedded Vision Systems

W. James MacLean
Department of Electrical & Computer Engineering,
University of Toronto, Toronto, Ontario, M5S 1A1
macLean@eecg.toronto.edu

Abstract

Reconfigurable hardware, in the form of Field Programmable Gate Arrays (FPGAs), is becoming increasingly attractive for digital signal processing problems, including image processing and computer vision tasks. The ability to exploit the parallelism often found in these problems, as well as the ability to support different modes of operation on a single hardware substrate, gives these devices a particular advantage over fixed architecture devices such as serial CPUs and DSPs. Further, development times are substantially shorter than dedicated hardware in the form of Application Specific ICs (ASICs), and small changes to a design can be prototyped in a matter of hours. On the other hand, designing with FPGAs still requires expertise beyond that found in many vision labs today.

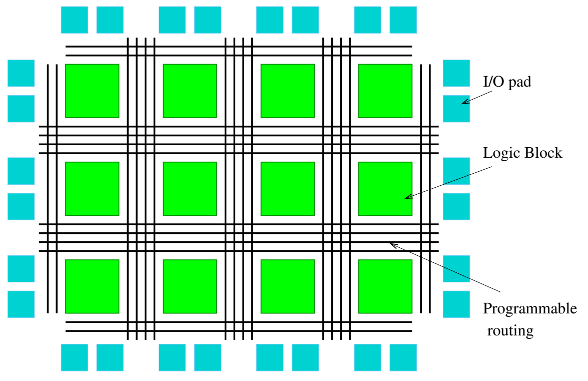
This paper looks at the advantages and disadvantages of FPGA technology, its suitability for image processing and computer vision tasks, and attempts to suggest some directions for the future.

1. Introduction

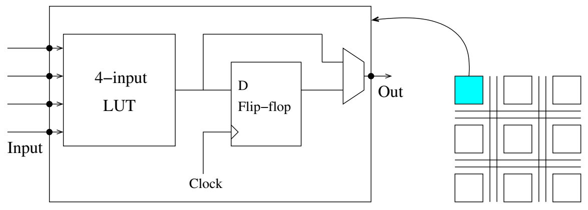
Perhaps motivated by the high computational complexity of many computer vision algorithms, there have been many attempts to create hardware implementations to achieve high-performance vision systems. The target applications have ranged from localization and pose estimation of parts in manufacturing settings, to biometric identification in banking applications, and depth estimation and target tracking for navigation systems and robotic control. Hardware implementations have been of three main varieties; DSP with special interfaces for video and other sensors, Application Specific ICs (ASICs, or sometimes “custom ICs”), or implementations based on reconfigurable hardware such as Field Programmable Gate Arrays (FPGAs). The latter technology has matured considerably over the past decade as chip densities and die sizes have increased, and as special features and development tools have simplified the task of designing for FPGAs. Certainly FPGA devices have been designed with embedded systems in mind, and are currently targeted at the automotive, communications and other industries: here the advantage is being able to design a wide range of devices with a small stock of actual devices, and to be able to refine and upgrade the designs without substantial time and financial burden. It is the intent of this paper to examine the current state of this technology, and assess its viability for implementing computer vision systems. Further, future directions will be suggested in order to develop a solid basis for such implementations to advance.

1.1. Previous Work

Perhaps the most common computer vision algorithm implemented using FPGAs is that of stereo disparity estimation [6, 28, 19, 30, 8, 13]. The PARTS reconfigurable computer [28] consists of a 4×4 array of mesh-connected FPGAs with a maximum total number of about 35,000 4-input LUTs. A stereo system was developed on PARTS based on the census transform, which mainly consists of bit-wise comparisons and additions [30]. Kanade *et al.* [19] describe a hybrid system using C40 digital signal processors together with programmable logic devices (PLDs, similar to FPGAs) mounted on boards in a VME-bus backplane. The system, which the authors do not claim to be reconfigurable, implements a sum-of-absolute-differences along predetermined epipolar geometry to generate 5-bit disparity estimates at frame-rate. In Faugeras *et al.* [8], a 4×4 matrix of small FPGAs is used to perform the cross-correlation of two 256×256 images in 140 ms. In Hou *et al.* [13], a combination of FPGA and Digital Signal Processors (DSPs) is used to perform edge-based stereo vision. Their approach uses FPGAs to perform low level tasks like edge detection and uses DSPs for higher level integration tasks. In [6] a development system based on four Xilinx XCV2000E devices is used to implement a dense, multi-scale, multi-orientation, phase-correlation based stereo algorithm [9] running at frame-rates. In [7], optic flow estimation (using Lucas & Kanade’s flow algorithm) is implemented on an FPGA system. It is worth noting that not all previous hardware approaches have been based on reconfigurable devices. An ASIC-based design, the Acadia



(a)



(b)

Figure 1: (a) Architecture of a generic FPGA. All FPGAs include the basic elements shown here. Newer FPGAs may also include embedded memory blocks, dedicated multiplier blocks, and even processor cores. (b) Simplified architecture of a logic block showing one, four-input lookup table (LUT). Each LUT can synthesize any four-input logic function, and may also be specialized for implementation of memory and shift registers. Newer FPGAs may contain as eight or more LUTs in a single logic block.

Vision Processor, is reported in [27, 3]; this system performs a number of "front-end" vision processing tasks such as stereo and motion estimation, and feature enhancement using either Gaussian or Laplacian image pyramids. Another system, the SRI Small Vision Module, is described in [21]; this DSP-based system is dedicated to stereo disparity estimation. An interesting trend worth noting is the use of commodity graphics hardware to implement stereo disparity estimation [29] and a general purpose vision processor, with projective flow as a sample application [10].

There are certainly more hardware-based, and even FPGA-based, implementations of vision algorithms than there is space to list here. The important thing to note is that numerous attempts have been made to use FPGA in implementations, yet there is no sense yet of wide-spread acceptance of this technology in the vision community.

2. Basic FPGA Architecture

A *Field-Programmable Gate Array* is a device containing logic circuitry whose function and interconnectivity can be altered by downloading a *bit-stream* that describes the desired connection/configuration pattern. This allows a user to *program* the chip by interconnecting the logic in a manner that achieves the desired functionality. The functionality can be changed by downloading a new bit-stream, allowing the same device to be re-used for different applications. All FPGAs consist of three major components [2]: 1) Logic blocks; 2) I/O blocks; and 3) programmable routing, as shown in Figure 1 (a). A logic block (LB) is a local collection of logic circuits, and can be programmed to perform a desired operation via the bit-stream. To implement a circuit on an FPGA, each logic block is programmed to perform a small part of the total logic required by the circuit and each I/O block is programmed to act as an input or output, as required. The programmable routing is also configured to make the necessary connections between logic blocks, and from logic blocks to I/O blocks.

The processing power of an FPGA is directly proportional to the processing capabilities of its logic blocks and the total number of logic blocks available in the array. Currently, most commercial FPGAs use logic blocks that contain one or more Lookup Tables (LUTs), typically with at least four-inputs. A four-input LUT can implement any binary function of four logic inputs. Figure 1 (b) shows the architecture of a simple logic block containing one four-input LUT and one flip-flop for storage. Logic blocks are sometimes designed to allow them to be efficiently used as local memory and shift registers, and they generally contain dedicated carry circuitry in order to simplify implementation of adders and multipliers.

The two leading manufacturers of FPGA devices are Altera Corporation [5] and Xilinx Inc. [15]. A look at their respective websites will give details on the current state-of-the-art for FPGAs, the Stratix-II^{TM} family from Altera and the Virtex-4^{TM} family from Xilinx. These devices, in addition to the features described above, have dedicated circuitry to assist in DSP applications (such as special multiplier blocks), hard-processor cores (*e.g.* PowerPC on the Virtex-4 family) and soft-processor cores (*e.g.* NIOS on the Stratix-II family), large amounts of on-chip memory, dedicated interface circuitry to simplify access to off-chip memory, and high-speed serial and parallel I/O capabilities. Table 1 gives a brief comparison between the top-end devices in the two families.

The amount of logic circuitry along with the other features of modern FPGAs makes programmable hardware a viable and efficient solution for accelerating complex image-processing and computer vision applications. For designs that are too large to fit on a single FPGA, a group of FPGAs connected with a programmable interconnection network can be used.

It should be pointed out that implementing floating-point arithmetic operations on FPGAs is very expensive in terms of the number of logic elements required. Therefore it is of-

	Stratix-II	**Virtex-4**
Logic Elements	Up to 179,400 Logic Elements	Up to 200,000 Logic Cells
Embedded Memory	Up to 9 Mb	Up to 10 Mb
Clock Speed	250 MHz (500 MHz internally)	500 MHz
DSP Support	Up to 96 per device: dedicated multipliers (8 9×9 or 4 18×18 or 1 36×36 per block) configurable for multiply & accumulate Clocks up to 420 MHz	Up to 512 Xtreme DSP slices dedicated multipliers (18×18, two's complement) configurable modes for multiply & accumulate Clocks up to 500 MHz
DSP Development	DSP Builder$^{\text{TM}}$	System Generator$^{\text{TM}}$
Processor Cores	NIOS (soft)	PowerPC 32-Bit (hard) ×2
Other	HardCopy$^{\text{TM}}$ to facilitate migration to ASIC	EasyPath$^{\text{TM}}$ to facilitate mass production of low-cost, pre-programmed FPGA devices, "on-the-fly" partial reconfigurability

Table 1: A comparison of features between Altera's Stratix-II family and Xilinx's Virtex-4 family.

ten necessary to work with fixed-point representations when implementing algorithms. Also, operations such as multiplication, and even more so division, can be costly if used too freely. Careful design is needed to minimize the number of these operations.

The use of floating-point will undoubtedly increase as the capacity of FPGA devices increases. The advantage of floating-point becomes apparent when operating on data with large dynamic range [11], although often filters and other signal processing primitives need only operate on very limited ranges. Gaffar *et al.* [11] compare fixed- and floating-point for a number of applications, including an FIR filter, a discrete cosine transform[1] (DCT), and a ray tracing computation. For the FIR filter their analysis shows that floating-point only becomes more efficient for a dynamic range of 10^{12} or greater, and in the case of the DCT they report that the fixed-point implementation uses only 10% of the number of LUTs and flip-flops as the equivalent floating-point implementation, although the latter consumed fewer embedded multiplier blocks. In [18] the authors describe parameterizable IEEE-compliant floating-point adders and multipliers, done in a high-level hardware language to make them portable across different FPGA architectures. The designs, being parameterizable, can be used in different applications to provide the desired level of precision with minimal resource utilization. In [7] a custom floating-point unit is implemented to efficiently provide a division operation for the final stages of a Lucas-Kanade optic flow implementation. The authors cite the dynamic range of the data at the point of division as being large enough to make the use of floating-point more efficient than fixed-point. Finally, Underwood [26] claims that peak floating-point performance is growing faster on FPGAs than it is on CPUs, suggesting that there may come a time when floating-point operations on FPGAs are not only common, but even preferred over CPU implementations in some high-performance computing applications. For the remainder of this paper the discussion will assume that most vision algorithm implementations using FPGAs will use predominantly fixed-point arithmetic.

[1]The DCT is a core part of JPEG image compression, for example.

2.1. FPGA/DSP Design Tools

Design tools to assist DSP design with FPGAs fall into two main categories: GUI development environments that integrate with Matlab's Simulink, and Intellectual Property (IP) libraries. The development environments, DSP Builder$^{\text{TM}}$ for Altera and System Generator$^{\text{TM}}$ for Xilinx, allow the designer to select glyphs representing functional modules from libraries and graphically interconnect them to achieve the desired functionality. The glyphs provide the means to parameterize the functions, for example the number of data points to be used in an FFT. Since fixed-point arithmetic is standard, modules to convert between different precisions are included in the libraries. Library modules are as simple as up/down-sampling blocks or FIR filter blocks or complex arithmetic, and as sophisticated as Reed-Solomon or Viterbi coding/decoding blocks or numerically-controlled oscillators. Both environments provide support for multi-clockrate designs, allow for direct inclusion of supported FPGA hardware boards in the simulation loop to allow the designer to simulate directly on the target FPGA device as soon as possible, and they allow for "bit-true/cycle-true" simulation of the design, meaning that it should be possible to know the state of every signal for every clock cycle during design simulation. The user can write their own custom blocks directly in VHDL or Verilog for inclusion in designs[2]. Most of the IP modules (or "cores") are aimed at the communications industry, and only very simple image-processing

[2]At present, such blocks cannot be simulated in Matlab.

operations such as basic edge detection are currently supported. Higher-level computer-vision cores are available from third-party suppliers, for example SBS Technologies [14].

3. Evaluation

In this section, the advantages and disadvantages of FPGAs for computer vision implementations will be discussed.

3.1. Advantages

The obvious main advantage of FPGA-based design is the flexibility to exploit the inherently parallel nature of many vision problems. For example, many vision algorithms require the repeated application of the same local operation, such as application of a convolution mask, to every region in an image. In a serial processor this can be quite time consuming, but in an FPGA multiple convolutions can be taking place simultaneously. Compared to ASIC designs, the design-implement-test-debug cycle can take place on the order of hours and not months, and making small modifications to an existing design is a relatively simple task. Also in comparison to ASIC hardware, less actual hardware is needed if the system is to support multiple, mutually-exclusive modes of operations. In the FPGA design the system can be reconfigured in a matter of milliseconds whereas separate hardware is required for each aspect of the ASIC implementation. In this sense the FPGA design is like software in that it mimics the ease with which one program can be stopped and another started. However, FPGAs have shown greater throughput in many problems, and can go places fast serial processors cannot (for example, radiation considerations may prevent fast CPUs from being usable in space robotics, whereas the relatively slower clock speed of FPGAs makes them less susceptible to such problems. Here the added bonus of a smaller hardware footprint also gives FPGAs an advantage over ASICs).

Modern design tools, as described in Section 2.1, makes design using FPGAs closer to that of software than it has sever been in the past. Although the process of logic synthesis and signal routing from a high-level description may take several hours in the case of a large design, it is still fast and simple enough to allow significant progress to be made in relatively short time frames. Further, the software-engineering concepts of developing and testing modules stand-alone, and then assembling the modules into a final system and doing integration testing, is applicable to FPGA design. This style of development allows relatively shorter synthesis times, speeding up the process. The use of pre-packaged modules in a graphical design environment means that system design and testing may be carried out directly by the individuals knowledgeable about the algorithm to be implemented, and not just a specialized hardware engineer. It is also true that designing with FPGAs is much less expensive in terms of design tools and hardware test-beds, allowing smaller companies, and even individuals, to pursue design objectives that were previously only accessible to larger companies.

Finally, in the case of embedded systems intended for commercial use, modern FPGAs provide protection from reverse engineering through the use of encrypted configuration bit-streams.

3.2. Disadvantages

Perhaps the single biggest disadvantage to using FPGAs for any sort of digital signal processing is the practical requirement to implement algorithms using primarily fixed-point arithmetic. As was noted earlier, while FPGAs are capable of implementing floating-point arithmetic, in many cases this requires far too much of the available resources to be feasible, especially if the operation is being replicated many times to take advantage of the speed-up a parallel architecture can provide. Most vision research produces algorithms that rely on the availability of floating-point arithmetic. While it is always possible to convert such algorithms to fixed-point, it requires a labour-intensive analysis of the minimum precision required at each stage in the algorithm's data flow, and then a decision as to how to balance this precision against the available resources of the target FPGA devices [6]. What is worse, should the design later be re-targeted to a device with a different capacity, then the analysis may need to be re-done. It is probably true that many algorithm designers give little thought as to the required level of precision for the algorithms they design, and it only merits attention when the algorithm becomes numerically unstable even on a floating-point processor. On a more positive note, the attention paid to DSP applications by FPGA manufacturers may well mean that native support for some (limited) amount of native floating-point support may be a feature found on FPGAs in the not-too-distant future. DSP processors once made a similar migration from fixed- to floating-point. Another disadvantage of FPGA-based designs lies in the finite resources of target devices, alluded to above. Unlike a general purpose workstation, when you run out of resources on the target device, you are done. It is possible to design multi-FPGA systems to increase resource availability, but then the designer, at least at present, has to contend with the design and resource overhead of partitioning the algorithm and transferring intermediate results between the different FPGA devices. At present, this often requires more understanding of digital logic design than the high-level design tools would appear to require.

In a direct comparison to ASIC devices, ignoring the issue of cost and design time lines aside, implementations using FPGAs are typically less efficient due to the overhead on the device for the configuration circuitry, including I/O and the SRAM cells required to hold the current design. This leads to larger device sizes and larger power consumption. The ASIC design process also allows optimization of the circuitry to allow for faster clock speeds than found on FPGAs. This slower clock speed may be an advantage, as seen in the previous section.

The last major disadvantage is the design process itself. While it is getting easier all the time, it is still true that developing any system on an FPGA usually requires some savvy when it comes to digital design. Unlike software, where issues related to the target platform are now well hidden from the software developer, FPGA based design still requires some knowledge of the concepts of clock signals and propagation delays, and familiarity with logic elements such as FIFOs, registers, flip-flops, and the architecture of adders, multipliers and the like. The designer may need to be familiar with pipelined multipliers, for example, to understand the trade-off between the number of logic elements required to implement the multiplier and the number of clock cycles required for the multiplier to produce an answer. This may further bring up the issue of multi-clock designs, and converting between different clock rates at various points in the data pipeline. That being said, FPGA manufacturers are looking to expand potential markets for their products, and are keenly aware of the need to make the design process as simple and robust as possible.

3.3. Moore's Law?

Perhaps one of the most contentious issues in the debate over software- *vs.* FPGA-based implementations is that of *Moore's Law* [25], which states that the speed of computer systems tends to double roughly every 18 months. Why bother with hardware-based designs when we can wait a while until general purpose processors are fast enough to do the job? There are a number of responses to this viewpoint. First, if we can do something today with an FPGA, why wait? FPGA-based design is becoming easier (and thus faster) all the time, so often we can have what we want sooner. It is worth pointing out that many FPGA implementations offer speed-ups of one to two orders of magnitude. This would suggest that Moore's law could leave us waiting 5 to 10 years for a fast enough processor. Also, even if general purpose CPUs become fast enough to implement a particular algorithm, then as suggested in Section 3.1, there may be environments where an FPGA-based implementation is still more suitable. Finally, it should be pointed out that Moore's Law applies to FPGAs just as it applies to serial CPUs, so that when CPUs are fast enough to implement a certain algorithm, FPGAs will be capable of implementing even more computationally intensive algorithms. If Underwood [26] is correct, FPGAs may surpass traditional CPUs in performance for some types of computations.

4. Research Directions

Having weighed the advantages and disadvantages, and with a slightly optimistic outlook on the future, it is the author's opinion that FPGAs have an important role to play in the design of future embedded vision systems. This section will consider some of the obstacles that still need to be overcome in order to realize this potential.

Work needs to be done on the automated analysis of the required precisions for converting a given algorithm to fixed-point [6]. This should be tied to the type and number of target devices, and the problem of partitioning an implementation across these devices. Some work has already been done in this area [4, 11], with the latter presenting a system that computes appropriate bit-widths (as well as automatically selecting between fixed- and floating-point arithmetic) given C++ or System Generator files as input, although no mention is made of automatic partitioning of an algorithm across multiple FPGA devices, or even choosing bit-widths to accommodate the finite resources of a particular target device. It is very likely that this research will be undertaken by researchers in the FPGA and DSP community regardless of what happens in the vision community.

Related to the analysis of precision and algorithm partitioning is the issue of scalability of designs. More concretely, if you would like to implement a particular algorithm on a particular device, how do you answer the questions, "What is the best I can do with the available resources?", and given performance specifications "Are the given resources minimally sufficient?" As vision researchers will be implementing their own algorithms, they will have to deal with this issue. It is highly desirable that designs can be made scalable to fit different devices of different capacities, and be easily transportable across different families of devices. Again, guidance on this issue will come from FPGA experts.

Given the availability of embedded hard- and soft-processors, and given that there will always be aspects of vision systems that are more appropriately done in software, vision systems implemented on FPGAs will need to take advantage of embedded processors to optimize the performance and flexibility of the implementation. The correct balance will depend on the algorithm being implemented. Soft-processor cores are particularly of interest, since they can be customized for the task at hand, and can be instantiated multiple times in a single FPGA device.

Perhaps the biggest undertaking that will facilitate implementation of vision systems on FPGAs is the development of a library of interchangeable vision cores that could be used in high-level design environments. These cores, or modules, would need a model for direct communication between them. Such models exist, *e.g.* in the SIMPPL architecture [24], each module has a programmable controller to simplify (1) direct data communications between modules, and (2) modifying the module's internal functionality without changing the interface to adjoining modules. When data needs to be shared among multiple modules, then an architecture such as the *Open Core Protocol* (OCP) [1] simplifies connecting modules to buses by providing a standard interface to each module, and then different cores to connect to a variety of supported standard bus architectures. The obvious question then becomes, "What modules would you build?" A useful library would likely include modules for image acquisition from a variety of standard camera interfaces, multi-scale/pyramidal image representations, feature detectors (*e.g.* Harris corner features [12], SIFT features [22, 23], and other affine-invariant features), stereo modules, optic flow modules, particle filter frameworks, egomotion estimation, generic segmentation/clustering modules (to do colour- or motion-based segmentation).

An example of how such modules might work together is as follows: a feature detector can feed features to a module that segments point correspondences based on 3-D motion, where the latter is based on recovering R and $\vec{T}$ from an E-matrix egomotion module. The feature detector itself may rely on a multi-scale image representation module. Another example is the design of "smart-sensors", for example a sensor that outputs not only images but lists of tracked features to a general-purpose processor for higher-level analysis.

A longer term goal might be the study of "on-the-fly" reconfiguration of FPGA devices. In [20] an architecture is proposed for processing of multiple data-streams where each data stream can be initiated, terminated & reloaded without interruption of the other data streams. Such dynamic re-configuration may prove useful in clustering implementations that need to support splitting and merging of clusters, to give one example. Modern FPGA families support such reconfiguration for a variety of purposes, including recovering from configuration errors where a configuration bitstream becomes corrupt, and part of the device has to be re-configured to restore proper operation [17, 16].

5. Summary and Conclusions

This paper provides a brief overview of the current state-of-the-art of FPGA technology, including the available resources to facilitate design on these devices. The major advantages are reconfigurability, and the ability to exploit parallelism in vision problems thus leading to substantial performance improvements. The major disadvantages are conversion of algorithms to fixed-point representations, hard resource limitations and lack of portability of designs between devices[3], and the relatively more complicated design process as compared to software. The role of Moore's law in the debate about which technology to choose has been considered. In the final analysis FPGA technology looks promising, but a number of issues need to be addressed. These are: automation of analysis for fixed-point precisions required and the task of partitioning algorithms across multiple devices, taking advantage of hard/soft-processor cores to include software elements to the design, development of standard and interchangeable modules to facilitate building more complex systems, and possible research into systems that re-configure themselves as a normal part of their operation.

[3]This problem is rapidly disappearing as IEEE standards are adopted by third-party manufacturers of synthesis tools.

Acknowledgments

The author would like to thank the following for valuable discussions related to this work: James Clarke, Paul Chow, Ahmed Darabiha, Frank Dellaert, Frank Ferrie, Allan Jepson, Jim Little, Divyang Masrani and Jonathan Rose.

References

[1] VSIA homepage. Online: `http://www.vsia.org`.

[2] Steve Brown, R. Francis, Jonathan Rose, and Zvonko Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, May 1992.

[3] Peter J. Burt. A pyramid-based front-end processor for dynamic vision applications. *Proceedings of the IEEE*, 90(7):1188–1200, July 2002.

[4] M. L. Chang and Scott Hauck. Precis: A design-time precision analysis tool. In *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 229–283, April 2002.

[5] Altera Corporation. Altera homepage. `http://www.altera.com/`.

[6] Ahmad Darabiha, Jonathan Rose, and W. James MacLean. Video-rate stereo depth measurement on programmable hardware. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision & Pattern Recognition*, volume 1, pages 203–210, Madison, WI, June 2003.

[7] Javier Díaz, E. Ros, S. Mota, F. Pelayo, and E. M. Ortigosa. Real-time optical flow computation using FPGAs. In *Proceedings of the Early Cognitive Vision Workshop*, Isle of Skye, Scotland, June 2004.

[8] Olivier Faugeras, Bernard Hotz, Hervé Mathieu, Thierry Viéville, Zhengyou Zhang, Pascal Fua, Eric Théron, Laurent Moll, Gérard Berry, Jean Vuillemin, Patrice Bertin, and Catherine Proy. Real time correlation-based stereo: Algorithm, implementations and applications. Technical Report Research Report 2013, INRIA Sophia Antipolis, August 1993.

[9] David J. Fleet. Disparity from local weighted phase correlation. In *International Conference on Systems, Man and Cybernetics*, volume 1, pages 48–54, 1994.

[10] James Fung and Steve Mann. Using multiple graphics cards as a general purpose parallel computer: Applications to computer vision. In *Proceedings of the 17th International Conference on Pattern Recognition*, pages 805–808, Cambridge, UK, August 2004. Online: `http://www.eyetap.org/papers/docs/procicpr2004/`.

[11] Altaf Abdul Gaffar, Oskar Mencer, Wayne Luk, and Peter Y. K. Cheung. Unifying bit-width optimisation for fixed-point and floating-point designs. In *12th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004)*, pages 79–88, Napa, CA, April 2004.

[12] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings Fourth Alvey Vision Conference*, pages 147–151, Manchester, United Kingdom, 1988.

[13] K. M. Hou and A. Belloum. A reconfigurable and flexible parallel 3d vision system for a mobile robot. In *IEEE Workshop on Computer Architecture for Machine Perception*, New Orleans, Louisiana, December 1993.

[14] SBS Technologies Inc. SBS technologies homepage. `http://www.sbs.com/`.

[15] Xilinx Inc. Xilinx homepage. `http://www.xilinx.com/`.

[16] Xilinx Inc. Xapp216 v1.0: Correcting single-event upsets through virtex partial configuration, June 2000.

[17] Xilinx Inc. Xapp151 v1.6: Virtex series configuration architecture user guide, March 2003.

[18] Allan Jaenicke and Wayne Luk. Parametrised floating-point arithmetic on FPGAs. In *IEEE Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 897–900, May 2001.

[19] Takeo Kanade, Atsushi Yoshida, Kazuo Oda, Hiroshi Kano, and Masaya Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *Proceedings of the 15th IEEE Computer Vision & Pattern Recognition Conference*, pages 196–202, San Francisco, June 1996.

[20] Lev Kirischian, Irina Terterian, Pil Woo Chun, and Vadim Geurkov. Re-configurable parallel stream processor with self-assembling and self-restorable micro-architecture. In *Proceedings of International Conference PARELEC-2004*, Dresden , Germany, September 7-10 2004.

[21] Kurt Konolige. Small vision systems: Hardware and implentation. In *Proceedings of the Eighth International Symposium on Robotics Research (Robotics Research 8)*, pages 203–212, Hayama, Japan, October 1997.

[22] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh International Conference on Computer Vision*, pages 1150–1157, Kerkyra, Greece, 1999.

[23] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:90–110, 2004.

[24] Lesley Shannon and Paul Chow. Simplifying the Integration of Processing Elements in Computing Systems using a Programmable Controller. In *IEEE Symposium on Field-Programmable Custom Computer Machines (FCCM'05)*, April 2005. To appear, `http://www.eecg.toronto.edu/~pc/research/publications/shannon.fccm2005-submitted.pdf`

[25] Ilkka Tuomi. The lives and deaths of Moore's law. http://www.firstmonday.dk/issues/issue7_11/tuomi/, 2002.

[26] Keith D. Underwood. Fpgas vs. cpus: Trends in peak floating-point performance. In *12TH ACM International Symposium on Field Programmable Gate Arrays (FPGA 2004)*, pages 171–180, February 2004.

[27] G. van der Wal and P. Burt. A VLSI pyramid chip for multiresolution image analysis. *Int. Journal of Computer Vision*, 8:177–190, 1992.

[28] J. Woodfill and B. Von Herzen. Real time stereo vision on the parts reconfigurable computer. In *5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 201–210, 1997.

[29] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Proceedings of the 2003 IEEE Conference on Computer Vision and Pattern Recognition*, pages 211–218, Madison, Wisconsin, June 2003.

[30] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proceedings of the 3rd European Conference on Computer Vision*, pages 150–158, May 1994. http://www.cs.cornell.edu/rdz/Papers/Archive/neccv.ps, http://www.cs.cornell.edu/rdz/Papers/Archive/nplt-journal.ps.gz.