Reconfigurable Processor Architectures

Ian Page,
Oxford University Computing Laboratory,
Parks Road, Oxford OX1 3QD, U.K.

Submitted for a Special-Issue of "Microprocessors and Microsystems"
on Hardware/Software Codesign

Abstract

No particular application is well-supported by a conventional microprocessor which has a predetermined set of functional units. This is particularly true in highly dynamic areas, such as multimedia, communications and other embedded systems. We suggest that additional silicon is used to provide hardware which can be dynamically configured to support *any* application. By combining a conventional microprocessor and FPGA reconfigurable logic on one chip, commodity pricing is maintained and yet the same part can effectively support a wide range of applications. A novel FPGA architecture is outlined which is particularly suitable for this style of implementation.

Keywords : FPGA, computer architecture, parallel processing, embedded systems.

Introduction

Computer architecture has been a lively and relevant topic for research and development and much has changed over the fifty years or so of the history of modern computers; however, much has also remained the same. This paper explores the thesis that radical changes are set to affect all aspects of computer architecture which are at least as far-reaching as any that have been witnessed in the past half-century. The force behind the changes is the newly-emerging technology of dynamically reconfigurable hardware. Dynamically Programmable Gate Array (DPGA) chips are today's most potent implementations of such hardware. Their function can be changed in milliseconds under purely software control; they are early embodiments of truly general-purpose hardware. This technology offers the possibility of architectures which change *during operation* to support the current application as efficiently as possible.

These reconfigurable hardware components are already being used in combination with traditional processors to deliver novel ways of implementing applications. The very fact that a combination of a processor and some reconfigurable hardware is already so useful, is a direct pointer to a future in which reconfigurable hardware finds its way inside processors and radically changes their nature, what they can do, and the ways in which we design and program them.

A great deal of work has been reported on the benefits to be obtained by coupling microprocessors with Dynamically Programmable Gate Array (DPGA) components [5, 17, 16]. Our own work in this area was first reported in [22] where a modular system based on a closely-coupled 32-bit microprocessor and a DPGA was described, with fuller details appearing in [15]. We have demonstrated a number of applications running in this framework including pattern matching, spell-checking, video compression and decompression, video target tracking and others [21]. The success of these applications has convinced us that many applications can be run significantly faster when there is even a modest amount of reconfigurable hardware available to the host processor.

Significant increases in the speed of execution of applications are claimed despite the fact that circuitry implemented in DPGAs is nowhere near as fast or as dense as can be achieved with ASICs. DPGAs are however fast and dense *enough* to give significant support to some important applications, especially where an ASIC development would take too long or would be too costly. There is also another, slightly non-obvious, ameliorating factor in favour of DPGAs. Since they are easier to design and implement than ASICs, it should be the case that the time-to-market of a new DPGA is shorter than that of a new commodity microprocessor or ASIC. This means that using a newly-released DPGA gives a designer access to technology that is actually more up-to-date than an ASIC released at exactly the same time.

We believe that the key to exploiting the technological opportunity is the provision of appropriate software tools. We believe that most of the DPGAs that will be closely coupled to tomorrow's microprocessors will in fact be configured by programmers writing and compiling programs, rather than electronic engineers designing hardware. The crucial task is to remove the differences between hardware and software

design, so that it is no longer necessary to maintain two separate development teams for the hardware and software components of a new system. We argue that the current paradigms of hardware design are simply inappropriate for developing software, but that the paradigm of computer programming is in fact quite well-suited to the task of hardware design.

This paper sketches a number of related ideas and covers few of them in depth. As such, it is intended to contribute to the discussion on areas for future work and to help develop a taxonomy for this new class of computing device.

Why Processors and DPGAs Should Be Integrated

In the past, one of the responses to an increased availability of silicon area has been to provide additional functionality on commodity microprocessor chips. We have seen microprocessors grow from 4-bit to 64-bit, and single execution units to multiple execution units; we have seen the introduction of on-chip memories, DMA engines, communications links, caches, and floating-point co-processors. It is certain that more of this style of development is to come, but there is a problem with it that simply will not go away. Processors which exhibit this sort of 'creeping featurism' are clearly attempting to provide wide-spectrum support for applications. However, despite this attempt to provide high-performance general-purpose computing, they inevitably become less and less suited to any *particular* application because a decreasing fraction of the chip is going to be useful for that application.

In the final analysis, hardly anyone really wants a general-purpose computer. Indeed the entire software industry exists to provide programs with which users can turn their general-purpose computers into application-specific processors, be it for running a spreadsheet or a multimedia application. For the duration of any interaction with a software package, the user wants a machine which will run that application both quickly and cost-effectively. The 'general-purpose' nature of the underlying compute engine is of no direct relevance to the user. It is simply historical fact that the best way of supporting a wide range of applications has been to use general-purpose computers. It has always been possible, though very costly, to build application-specific processors. However, we believe that coupling DPGAs with microprocessors and appropriate hardware compilation technology will usher in an era where such application-specific processors can be created in milliseconds in response to the changing demands on a system.

There is an often-repeated argument which says that there is no point in designing application-specific processors, because general-purpose processors are getting fast and cheap enough to support any application. This is simply not true in general. The very fact that there is an increasing market for chips which support high-speed graphics operations, video, multimedia, and communications shows that the general-purpose computer has been found lacking in these special-purpose applications.

Unfortunately, every application, or application type, will naturally require a different set of functional units to support it effectively. For any particular application, it may well be most cost-effective to add functional units which are relevant to that application; this results in highly specific chips which support the application extremely well, but they will not sell in large numbers and thus will suffer from high cost in comparison to commodity parts. However, when a commodity product has to support a wide range of applications, it is simply not cost-effective, after a certain point, to add more functional units when these have a fixed functionality. The result of doing so is a high-cost commodity part which is not well-matched to any application and where the utilisation of these units is low when any particular application is being executed.

In summary, we argue that the different ways to provide additional support for applications from commodity parts in the future are broadly the following:

Produce a wider range of application-specific processors: This route will undoubtedly be followed when the market for a particular application-specific processor is large enough. However, the design of these chips is a lengthy and expensive process and volumes may not normally be sufficient that their pricing can compete with commodity chips. Also, it does not suit the introduction of novel products or services where the market has to be built (necessarily slowly) on the introduction of a new product.

Add more fixed-function units: Adding more fixed features to a processor will usually mean that for

any particular application, the proportion of the new features actually used will decrease. With this model, we can look towards a future where more and more expensive silicon lies idle most of the time. Fixed-function features are a poor use of additional silicon unless they are going to be useful much of the time.

Add variable-function units: This solution offers a way out of the dilemma posed by the mismatch between the technology 'push' and the market 'pull'. General-purpose, reconfigurable hardware can deliver usable functionality across the whole range of applications. Advances in VLSI technology are easily exploited since reconfigurable hardware is fairly straightforward to design and it scales easily. Using programmable logic allows (i) many design decisions to be deferred so that near-market influences can be incorporated, (ii) fast introduction of new products, (iii) in-service upgrade of products, and (iv) more of the development becomes a controllable, software process.

Using the favoured third strategy, increasingly large amounts of silicon area can be devoted to general-purpose hardware which can be deployed in any way in order to support particular applications. Whether this happens in the factory by customising a volume chip for a particular product, or whether it happens dynamically in response to changing real-time demands on a delivered system is irrelevant; the same technological problems need to be solved to make the development of such systems a practical possibility.

The First Step : Connecting a Processor and DPGA

Because of the design of today's microprocessors, there are only a few ways to connect a DPGA to a microprocessor chip. The major interface to most microprocessors is a memory bus, consisting of sections for the parallel transfer of data, addresses and control information. By mapping device registers into the address space of the processor, the same bus structure is also conventionally used to provide device interfaces for input and output. A few processor chips have more than one bus, or may sport a number of DMA-controlled communication channels. While these structures can result in increased bandwidth, lowered latency and lack of interference between separate processes, they seem to present no new opportunities for DPGA interaction which are not already presented by the single multi-purpose processor bus, so we consider them no further here.

To connect the DPGA to a processor bus, we clearly have full generality if the entire bus is available at the pins of the DPGA, although it is perfectly reasonable to restrict the bus signals available to the DPGA for particular applications. The DPGA will need to be configured, and this is most effectively done under control of the microprocessor. The details of the reconfiguration operation are not particularly relevant to this discussion, so we simply assume that the programming port of the DPGA is mapped into the address space of the microprocessor and that software or a DMA process is responsible for loading configuration information.

An example of a system with a tightly-coupled microprocessor and DPGA is shown in Fig. 1. many other FPGA co-processor machines have been constructed such as [3, 23, 1] and there is a very useful list of such machines [7]. This is a block diagram of our HARP reconfigurable computing platform [14]. It consists of a 32-bit RISC microprocessor (a T805 transputer) with 4 Mbytes of DRAM, closely coupled with a Xilinx [27] DPGA-based processing system (XC3195A) with its own local memory. The microprocessor can load an arbitrary hardware configuration into the DPGA via its bus. A processor-controlled, 100MHz frequency synthesiser allows the DPGA to be clocked at its highest rate, which depends on the depth of combinational hardware and DPGA wiring delays inherent in a particular configuration.

HARP is a 90mm $\times$ 112mm printed circuit board using surface mount technology and is being made available on a commercial basis [26]. It is an industry-standard TRAM module (size 4) [12] and can thus be integrated easily with a wide range of off-the-shelf hardware and host computing systems.

The main input/output channels for the board are the 4 $\times$ 20Mbit/sec serial links supplied by the microprocessor. These make it very easy to link these boards together or to link them with other available TRAM modules, such as A/D converters, frame grabbers and other microprocessors and DSP systems.

The frequency synthesiser can generate an arbitrary clock for the DPGA circuit. The advantage is that the two processors can then run at different speeds, in accordance with their technology and precise configuration. The disadvantage of separate clocking is that the two processors are asynchronous. This

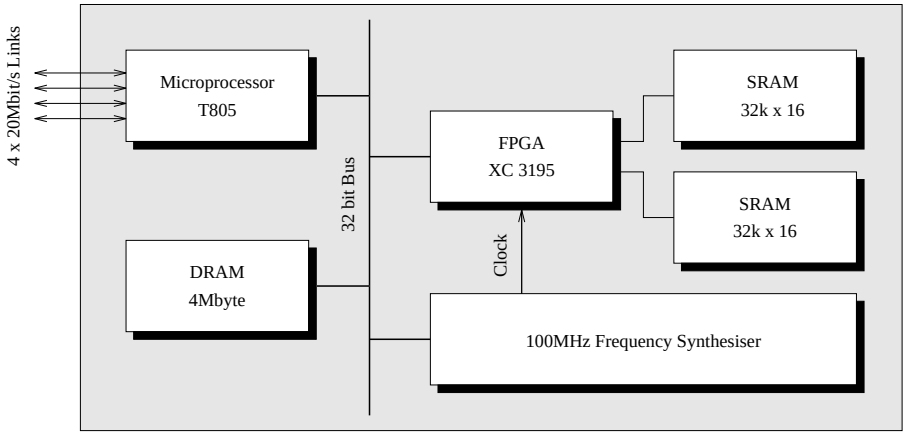

Figure 1: The HARP reconfigurable computer module

complicates the exchange of data and metastability problems have to be anticipated and dealt with in the interface.

Some system simplification can be obtained by clocking the DPGA circuit from the microprocessor clock. With this strategy, it may be possible to simplify the transfer of data between the processor and DPGA. Unfortunately, it can sometimes be difficult to ensure proper synchronous operation between the processor and a memory-mapped device since modern microprocessors may not make their internal clock signals visible beyond the chip boundary, and the activity on the bus is increasing divorced from the operations of the CPU itself.

The HARP system goes a little beyond the simple connection of an DPGA to a microprocessor bus in that it provides a two banks of SRAM connected to the DPGA. This memory is there as a general-purpose high-speed memory for storage of data or temporary results associated with the co-processor computation. By altering the DPGA configuration, this memory can be used privately by the DPGA algorithm, or it can be mapped onto the microprocessor bus where both processors can access it.

Our system is currently programmed by writing two programs in different, but closely related, languages. The software part is usually written in the occam [11] language, and the part destined for hardware is written in Handel [22, 24]. Handel is essentially a subset of occam which has been slightly extended to support arbitrary width variables. Both languages are closely related to the CSP language [9] which gives the secure intellectual foundation for the work and which provides a transformational algebra which allows programs to be routinely converted from one form to another [10]. Indeed the hardware compilation step itself can be expressed as a program transformation which starts with the user program and delivers another program, a so-called 'normal form' program, as its output [8, 4]. Both programs are in the same language, but the output program is in a very special form which can be directly interpreted as a hardware netlist. By this mechanism it is possible to develop both hardware and software within the same framework and many of the problems of relating separate hardware and software systems together simply disappear.

Models for DPGA Co-processor Operation

If we assume the bus-based close-coupling between a commodity microprocessor and a commodity DPGA as described above, there are a number of ways of using the system. It should be admitted that there may be no easy way to tell these modes apart as they are really just convenient ways of talking about different levels of interaction between the processors:

Adding new op-codes to the processor : The microprocessor issues a co-processor instruction, or some other instruction or instruction sequence, which can be detected by the DPGA and interpreted as a command-plus-data. If the DPGA algorithm is very fast, then it may be allowable to hold up the processor until a result is generated, otherwise the processor can continue operations, but some explicit synchronisation will be required when it requires the results from the DPGA.

Remote procedure call : The microprocessor issues an instruction or instruction sequence, which is interpreted by the DPGA as a remote procedure call (RPC) plus parameters. It is similar in effect to the above, except that it is a more 'heavyweight' interface and it is likely that explicit synchronisation will always be necessary. With a true multi-tasking system, such as that provided by an occam system running on a transputer, it is easy to arrange that the traditional procedure call semantics are maintained by the RPC, since the process which issued the RPC can be suspended, rather than the whole processor.

Client-server model : This model has the DPGA algorithm as a server process which makes it similar to the RPC mechanism, but where communications could be from any of the processes running on the microprocessor and the server process must arbitrate and prioritise between multiple near-simultaneous activations by the microprocessor.

Additionally, it is not unreasonable to consider the RPC and client-server models operating with the DPGA algorithm being 'the master'. Ultimately, this may be a good way for a real-time system running in the DPGA to be able to off-load difficult but infrequently-executed tasks onto the microprocessor which may have much more time and space resources to deal with complex exceptional cases.

Parallel process : This scheme takes one further step to distance the operations of the processor and DPGA. The DPGA algorithm runs as a top-level parallel process and communication between the two processors can happen at any point as jointly agreed by them. There is no natural master-slave interpretation of the relationship between the processors; the (distributed) algorithm is in sole charge of the pattern of communications.

Models for DPGA Memory Architecture

The program running on the DPGA co-processor will normally require some amount of variable and temporary storage for its operation. Only rarely would the entire circuit consist of combinational logic alone. Three different memory models suggest themselves:

DPGA uses no external memory : In some circumstances, the algorithm embedded in the DPGA co-processor may be able to operate without external memory at all. This situation is only reasonable when the co-processor needs rather little state for its operation, as current DPGAs have relatively few registers available.

DPGA shares microprocessor memory : Since the DPGA and processor share the bus, the DPGA is able to use any memory attached to the bus. However, this involves time and space overheads in the DPGA to deal with the necessary memory access arbitration. It may also be necessary to build a row/column address controller into the DPGA if dynamic ram is used. It also likely that access to this memory will considerably slow the co-processor in any data-intensive computation; the microprocessor is better able to deal with a slow memory interface as it will often have some on-chip cache arrangements, which are not currently a part of DPGA technology.

A special case is to implement the shared RAM as a dual-ported memory. This reduces the contention on the microprocessor bus, which is likely to be the limiting factor in many applications.

DPGA has its own local memory : It is frequently possible, and desirable, to organise the DPGA algorithm so that it can access data very rapidly; perhaps consuming and creating a data value on every clock cycle. Taken together with the fact that there are no instruction fetches needed by a hardware implementation of an algorithm, it can be seen that a conventional microprocessor memory is not a good match for a real-time, DPGA-based algorithm. This would be better served by one or more local banks of SRAM to provide the necessary bandwidth. Smarter SRAMs and SDRAMs with large address counters (i.e. not just burst mode support) will also aid high bandwidth data exchange.

Models for DPGA Memory Operation

The local DPGA memory may be used in a number of different ways. It should be noted that these models for DPGA memory operation are not necessarily mutually exclusive:

Memory holds a complete data set : The local memory is large enough to hold a complete data set for the computation. For example, this might be a complete frame of video in both compressed and uncompressed forms for a video compression algorithm.

FIFO queue : The memory acts as FIFO buffering while exchanging data and results with the microprocessor. An example would be the storage of the complete region of support for a real-time FIR (Finite Impulse Response) filter.

Cache : The local memory may be run as a cache onto a larger application data structure held by the main processor. It may be managed by predictive data exchange, programmed (high-level) requests from the microprocessor, or by traditional cache-line-on-demand methods.

Models for DPGA Program Execution

There are also a number of ways that programs may be embedded in the DPGA. These different models for supporting program execution in a DPGA can exploit different parts of the cost-performance spectrum of implementations.

Pure hardware : The algorithm is converted (by hardware compilation) into a hardware description which is loaded into the DPGA. This is the root technology on which everything else is built [22, 18].

Application-specific microprocessor : The algorithm is compiled into abstract machine code for an abstract processor, and the two are then co-optimised to produce a description of an application-specific microprocessor and a machine code program for it. The microprocessor description may then be compiled into a DPGA implementation. This option is explored in more depth later.

Sequential re-use of the DPGA : The algorithm may be too large to fit into the available DPGA, or there may be good engineering or economic reasons to split the algorithm so that it runs partially using one DPGA configuration, and then continues using another. The gains reaped by sequential re-use of the DPGA hardware must of course be balanced against the time taken to re-configure the DPGA. The sequential combinator in the source language (i.e. the semi-colon in C) is an obvious structure boundary for such a re-loading operation, but the **if** and **case** constructs can also provide good points for such code splitting.

Multiple simultaneous use : If the DPGA resources are large enough, it may be possible to have a number of algorithms co-resident and for each of them to interact separately with the host processor. At current levels of technology, this almost certainly means there would have to be a number of DPGA chips available.

This style of use is still uncommon, but it is likely to develop. It will be important to develop methods of producing and running position-independent circuits, analogous to position-independent code in the software world. This is a much harder problem when the underlying resource is a two-dimensional piece of silicon, rather than the one-dimensional abstract memory of a conventional processor. However, this only increases the need to develop good strategies for dealing with the problem.

On demand usage : The opportunity clearly exists for complex systems to be built where all the hardware does not exist at the same time, but where the real-time demands of the system dictate what hardware should be built and what should be destroyed. Indeed this precisely what is going on each time that a DPGA is configured.

The scheme here though is to take this a step further to where there is a relatively large collection of circuits which could be loaded into DPGA, and where the exact mix of circuitry at any point in time

depends on the actual activity of the system. There is a reasonable analogy here with virtual memory systems, so we can refer to this as 'virtual hardware'. Although it may seem a rather futuristic scenario, there are good reasons for believing that in the fields of multimedia, communications, databases and cryptography at least, the characteristics of the applications themselves are likely to demand this sort of highly flexible execution environment. Of course, this option can be combined usefully with all the others above.

The application-specific microprocessor

It is perhaps worth saying a little more about the application-specific microprocessor since it represents an essentially different implementation paradigm from the standard ones. A commodity microprocessor plus appropriate machine code is a common implementation paradigm for systems. It is cheap because of the commodity processor and memory parts, but it is also rather slow because the processor is *sequentially* re-using the expensive parts of the hardware such as ALU and registers. It seems reasonable to regard this paradigm as one end-stop in a spectrum of possible implementation strategies with different cost-performance characteristics, since it is hard to imagine any cheaper way of implementing complex systems.

At the other end of the spectrum are the ASIC, or highly *parallel* hardware, implementations produced by the hardware compilation techniques discussed here or by other means. These solutions tend to be expensive because of the large amounts of hardware and because of their non-commodity status, but they too can be regarded as an end-stop in the spectrum, since it is hard to think of any solution that would be faster in execution.

Having pegged out the ends of the spectrum it becomes interesting to see what might lie between them. There is naturally the possibility of a mixed-mode solution which is partly hardware and partly microprocessor-based, and such mixed-mode solutions will always be of interest. However, there is one essentially different paradigm which is an application-specific microprocessor (ASP) together with the appropriate machine code. This solution, where it is appropriate, inherits the key benefit of a microprocessor in that it offers fairly good performance and cheapness by rapid re-use of expensive hardware resources. But the ASP can also pick up some of the benefit of parallel hardware by having specialised instructions to support the application. Further details of this approach are provided in [2].

This has always been a possible solution for system designers, but it has rarely been available for reasons of cost. However, it is now possible to implement such microprocessors directly in DPGAs, and such processors can in fact be automatically designed as well [19, 20]. For example, we have built a 'processor compiler' which takes a user program, and compiles it to abstract machine code for an abstract processor. It then co-optimises both the processor and the machine code to produce a concrete machine code program together with a concrete processor description in the form of an ISP (Instruction Set Processor) program. This ISP program is then put through the hardware compiler to produce an actual application-specific processor. Although this process is automatic, the programmer can also have some influence over the final processor architecture by annotating parts of the original program. For example if the original program contains the statement `a := b + c*c`, then by simply writing this as `a := b + {c*c}` the programmer ensures that the automatically-designed microprocessor will have a squaring operation built-in as a single instruction.

This style of ASP implementation has yet to prove itself in practice, but it seems highly likely to do so. If it can in fact successfully inhabit a new point in the implementation spectrum, then there will be a natural follow-on, which is the 'processor construction kit on a chip'; alternatively, this is a DPGA with a number of special-purpose support circuits, depending on your point of view. This option is explored later in the paper.

How to Design Better DPGA Co-processors

The major step forward in terms of performance is to integrate the DPGA and the processor on the same die. We cover this topic later, so here we concentrate on a two-chip solution which is evolutionary rather than revolutionary.

The small size of DPGAs compared with the size of circuits we would like to implement with them is likely to remain a problem for the foreseeable future. Against this backdrop, we may be able to ameliorate

the problem by altering the internal architecture of the DPGA, its external interfaces, or by adding special-purpose functional units. The following sections briefly explore these strategies.

Changing the DPGA external interfaces

Here we assume that we are intending a DPGA to work cooperatively with a commodity microprocessor using one of the interfacing models described above. We look at what special-purpose circuitry can be added to the periphery of the DPGA in order to support processor-DPGA communication. The following list is a survey of the specialisations that might be attempted of a DPGA in order to make it work more effectively in the co-processor role.

Memory mapping : This supports the mapping of DPGA special-purpose hardware and general-purpose circuitry into the address space of the microprocessor, say with a configurable address decoder and a simple bus interface.

Memory interface : We add a fully configurable interface so that the DPGA can interface to various widths and styles of bus, while maintaining a simple logical internal interface which is presented to the DPGA core. A good example of this philosophy is the Xilinx 6200 design (formerly Algotronix Ltd.)[6, 13].

Data transfer : This goes beyond a simple synchronous memory interface to support handshaking, metastability guards, clock recovery etc.

Block transfer : Supports transfer of larger amounts of data by providing buffers, FIFOs, and counters.

DMA engine(s) : Supports the DPGA in accessing the memory of the host processor or of its own local memory. A DMA engine perhaps could be complex enough to support the following of linked-list data structures in memory, as many real-time embedded applications are based on such structures.

On-chip cache : As it becomes common for DPGA co-processors to range autonomously over large data sets in (private or shared) memory, it becomes necessary to support the memory interface with exactly the same sort of cache arrangements now common in microprocessors.

Synchronisation : It will be necessary at some points to ensure that the processor and DPGA can synchronise with each other, particularly when exchanging data. This could be mediated by the data transfer operations themselves by ensuring that the software and the DPGA implement end-to-end handshaking. On our HARP board we do this by implementing CSP-style channels between software and hardware. Special-purpose support could be added to support pure synchronisation operations.

Specialised Communication Interfaces : The DPGA could also support a limited number of specialised external interfaces, such as SCSI, ethernet, or ATM. Each of these possibilities would have to prove itself worthy of support, but it is at least conceivable that these and other protocols could be of sufficient use that a volume product was feasible.

Giving extensive support for particular high bandwidth communications protocols might be seen as changing the nature of the DPGA and force a split in the market. Such chips could easily be seen as a way of implementing an interface to some external network, in which the processor-specific interface was implemented in the DPGA and where some low-level processing by the DPGA might also relieve the host processor of an appreciable amount of processing. However, another view says that this is simply part of the future of the microprocessor itself, whether or not any DPGA technology is involved.

Adding special-purpose support structures

Future generations of DPGAs will almost certainly exhibit an increasing number of specialised structures where experience demonstrates that their addition offers worthwhile support across a broad range of applications and where it does not unduly affect the part's nature as a commodity item. Here we only

comment on some possibilities that might pay their way in the rather specialised world of DPGA co-processors.

An extension of all or part of the microprocessor bus, with appropriate buffering, into the heart of the DPGA is a way to ensure good communication across the DPGA boundary. There is an obvious problem that such a scheme demands that there is some reduction in flexibility about the way that these bus lines are brought to the DPGA pins. In particular, automatic place and route tools seem to have a very hard time when DPGA pins are heavily constrained. Though difficult, it should be worth tackling these problems head on, since the potential gain is considerable.

Following the route of extending the microprocessor bus into the DPGA leads quite naturally to supplying other specialised structures on this internal bus, such as multiplexors, ALU, and register structures which have good access to the bus. Naturally, it starts to beg the question of whether this component is then a traditional DPGA or a new sort of re-configurable microprocessor.

Novel DPGA Architecture

The architecture of DPGA cells and their routing networks is naturally a hot topic of research. Consequently, it is far from clear what the eventual winners will be in the race to set future standards for DPGA architectures. We do not attempt a general comparison of architectures here, but content ourselves with briefly describing a novel architecture of our own which may have something to contribute to the debate.

Starting with a survey of the desirable characteristics of DPGAs and a check list of what current DPGA architectures actually delivered, it was clear that there was a major gap in all current architectures for the support of wide gates and particularly the type of circuits which are best implemented with PLA, or And-Or plane, technology. As an example, a DPGA based on lookup tables cannot efficiently implement an address decoder for a microprocessor bus. However, PLAs (Programmable Logic Arrays) allow the efficient implementation of circuits with a large number of inputs and/or outputs.

We addressed this lack directly and developed and patented a novel DPGA which uses CAM (Content-Addressable Memory), rather than RAM (Random Access Memory) for logic implementation. Some conventional DPGAs employ small banks of RAM as lookup tables to implement simple Boolean functions. In our design, CAM allows us to implement all the types of circuit element supported by advanced RAM-based designs, including dense RAM structures, but in addition this novel architecture supports the implementation of arbitrary-sized PLA-style circuits. The architecture is described in more detail in [25].

PLAs are based on a plane of AND gates which process the inputs and pass the results to a plane of OR gates which then produce the desired outputs. Efficiency is improved since the intermediate AND-plane terms can be shared by many different outputs. It would clearly be desirable if a single DPGA design could be found which had the characteristics of both conventional DPGAs and also PLAs. We believe our design is the first effectively to address this issue.

Each DPGA block is inherently more flexible than a simple RAM lookup table since it has four inputs and *four outputs*. Four signals in each direction also provide local connections between the blocks. Thus, many interesting structures, such as RAMs, PLAs and decoders, can be built which pack densely into the DPGA while requiring no support from the global routing network. The inputs and outputs of the block are orthogonal, bidirectional, and to a large extent interchangeable. This gives the place and route software a great deal of easily exploitable freedom in the way that macros can be placed.

The DPGA Blocks in CAM Mode

The basic CAM cell is shown in Fig. 2 and has 4 vertical and 5 horizontal bus lines. It can be seen that this is similar in complexity to a static RAM cell, but the additional i/o lines allow the content addressability.

Our design uses an array of 4×4 of these one-bit CAM cells, directly abutted to each other, to build the CAM core of a single block of the DPGA. The block thus has a total of 4×4 vertical bus wires and 4×5 horizontal wires. The peripheral circuitry of one block processes and multiplexes these signals so that the 16-bit CAM block has 4 wires in each cardinal direction which directly join those of the nearest neighbour blocks.

The fact that there are four output wires from each block contrasts markedly with the single output of a lookup table as used in some DPGA architectures. This allows much greater density of user circuitry in

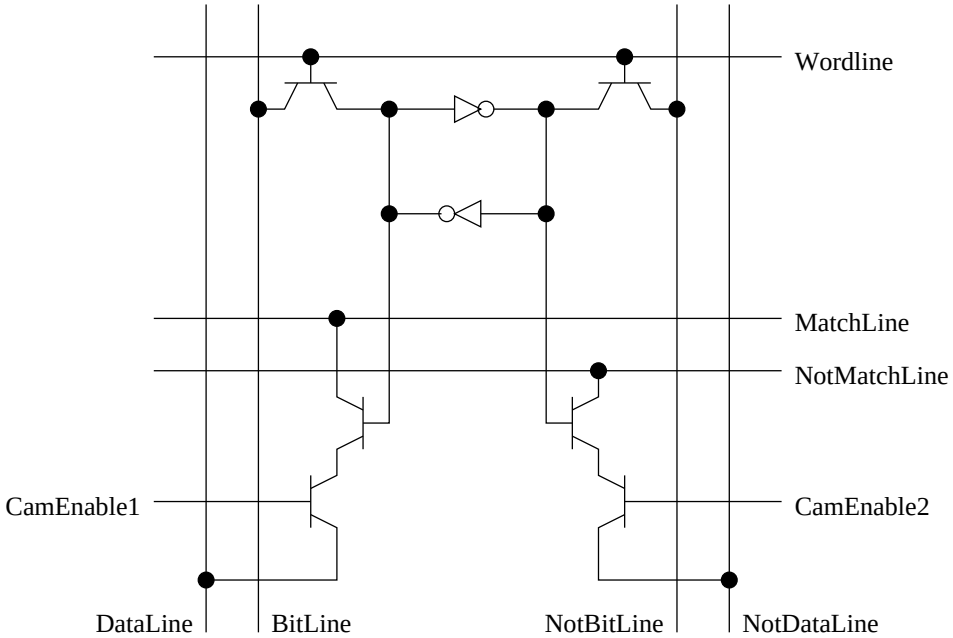

Figure 2: The CAM cell

the array for those fragments of the circuitry which can exploit it, and these fragments are precisely the ones which are poorly supported in the traditional architecture.

The CAM block is an ideal unit for building And-Or planes and it needs only modest support from the block's peripheral circuitry to multiplex and de-multiplex signals at plane boundaries. These blocks, operating in CAM mode, can be grouped to form arbitrarily large PLA structures as shown in Fig. 3. The blocks can also be used for routing and corner-turning as they can be programmed to act like crossbar switches.

The blocks can also be used efficiently to build CAM arrays of arbitrary size. CAMs have already proven to be useful building blocks in applications such as caches and we expect to find other uses of CAM in applications where previously they had not been considered as part of the solution space. This feature is a bonus directly attributable to the novel design of the CAM-DPGA, rather than an original design requirement.

The DPGA Block in RAM Mode

In RAM mode, each block can operate as a fully functional 16-bit lookup table. However, having four outputs, it can be used to generate one function of four variables, two functions of three variables, four functions of two variables etc. Fig. 4 shows how the CAM-DPGA blocks, operating in RAM mode, can be grouped to form arbitrarily large RAM structures.

The architecture rather neatly solves the twin problems of building dense RAM arrays and building the necessary decoder logic. The decoder blocks simply use the CAM mode with four horizontal outputs per block, and the RAM blocks use the RAM mode. The difference between RAM mode and CAM mode is simply the configuration of the peripheral circuitry within the block.

Combining Processor and DPGA on the Same Chip

If our predictions about the relevance of DPGA co-processors prove correct, then there will soon follow a commodity market for a DPGA and processor on the same chip. This will obviously yield a reduction in the number of parts in a typical system implementation, and it will also speed up communication between the two processors, and may also simplify the communication and clocking arrangements. By this stage

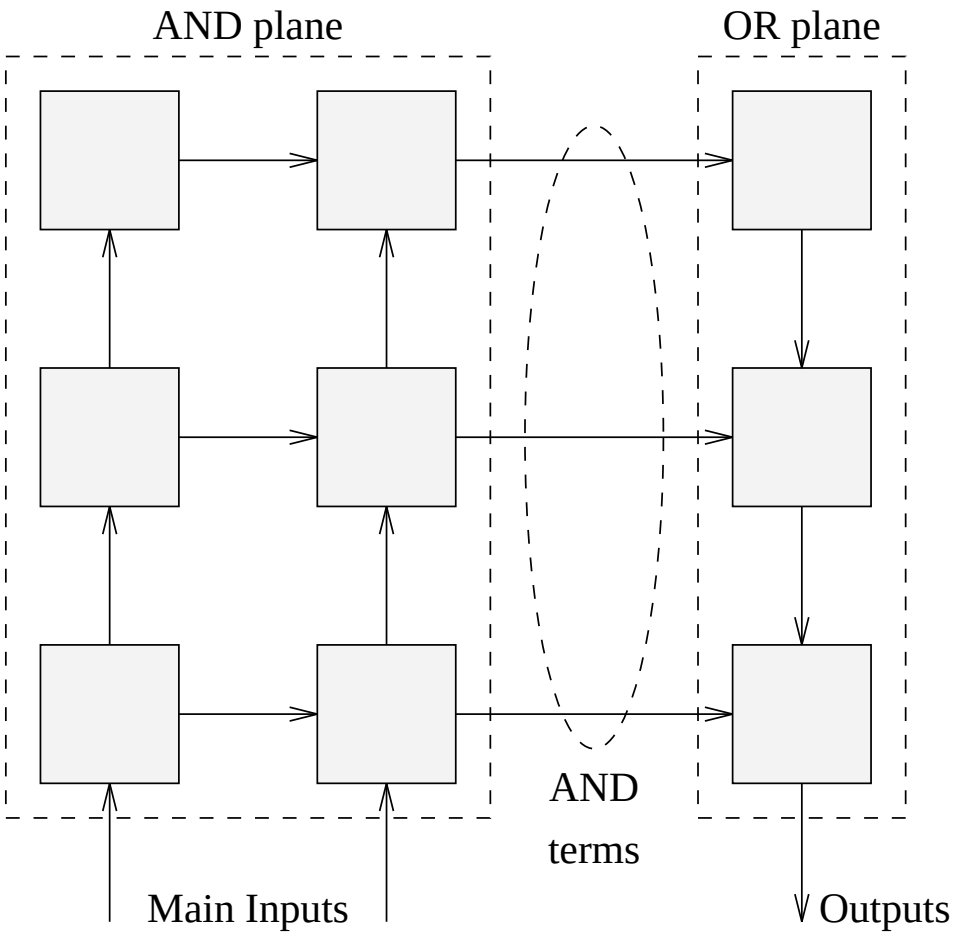

Figure 3: A larger PLA built from CAM-mode blocks

of evolution the problems of DPGA-processor interaction will have been well characterised across a wide range of real-world applications. Consequently, simply combining the two elements on a single chip is economically interesting but essentially straightforward in a technical sense, and will not be considered further here.

However, there is also scope for real technical innovation since there is no longer the same necessity to regard the processor core as sacrosanct and we can consider making changes to it as well as to the DPGA so that the combination becomes even better at supporting user applications. It may or may not be relevant to provide some measure of object code compatibility between the processor and its immediate forbears, but this is to a large extent a separable issue from what follows, so it will be ignored here.

The opportunity to be seized is nothing less than the complete re-appraisal of computer architecture itself, and to examine what role reconfigurable hardware has to play in the implementation of commodity microprocessors. We have already seen that complete microprocessors can be implemented in DPGA technology, but at a rather large cost since processors are highly structured objects and the DPGA is designed to support relatively unstructured circuits.

We have also suggested that a DPGA with a major datapath and some specialised components mounted on it would be a useful evolutionary step towards a better integration of the two technologies. One natural conclusion of this development would be the 'processor construction kit on a chip'. Another is the reconfigurable processor which is a processor with a relatively fixed high-level architecture, but with a significant amount of reconfiguration capability both within and between the specialised architectural components. These two concepts are related, but they are worth distinguishing as they result in significantly different products, and with significantly different software support costs.

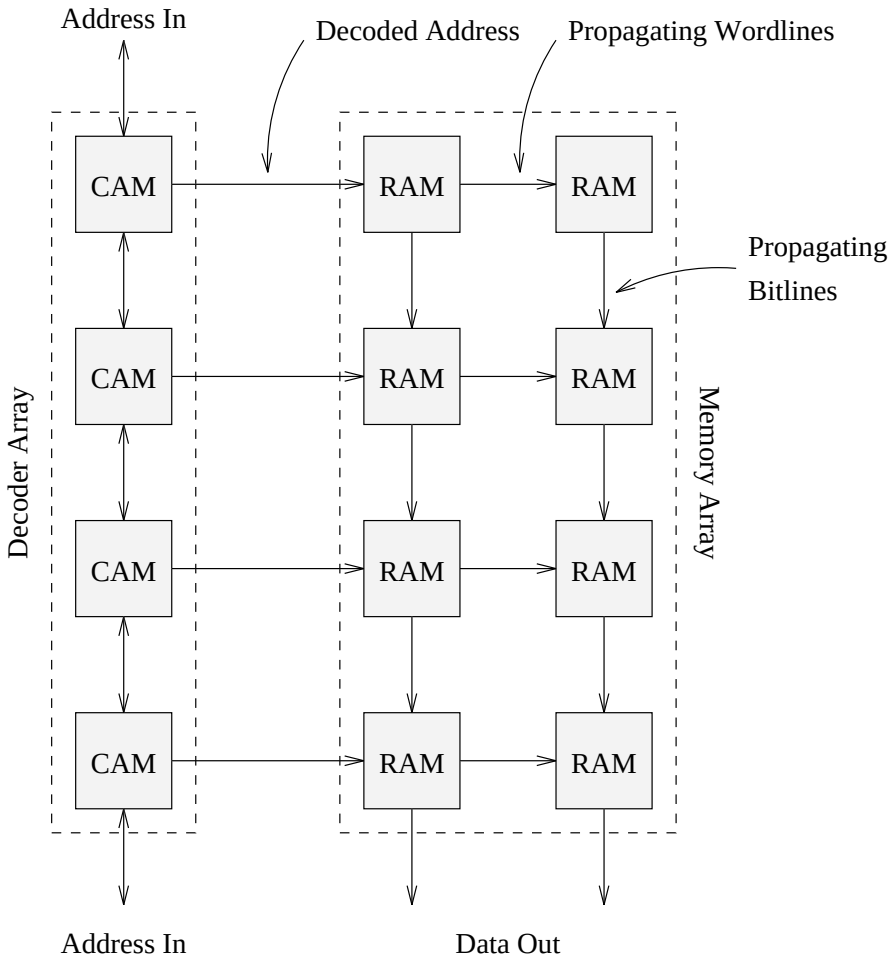

Figure 4: A RAM structure in CAM DPGA

The reconfigurable processor

The notion of the reconfigurable processor as presented here contains any microprocessor with a relatively fixed architecture which uses DPGA technology within one or more of its major architectural modules. The pairing of a microprocessor and a DPGA co-processor as presented earlier would certainly fall into this category. However, there are definite possibilities for incorporating DPGA resources in other parts of the architecture and this section reviews some of those which appear to have a useful purpose.

ALU : The ALU is an obvious place to put reconfigurable logic, however ALUs are already fairly well evolved for their standard purpose. The major opportunity seems to be to support special-purpose operations, and non-standard data formats. For instance, a 32-bit ALU could treat data as 4 $\times$ 8-bit bytes for an ordering operation, or a database application might use a dataset-specific field extraction and processing operation.

Register file : There is an opportunity to build special-purpose registers into an otherwise standard register file. Some registers might be constructed as accumulators or auto-incrementing registers, or assignment to them might side-effect some operation (perhaps input/output) elsewhere in the machine. If the application commonly uses data formats considerably smaller than the datapath width of the processor, then the size of the register file can be made smaller, or alternatively the number of registers can be increased, by implementing short registers. Addressing of the register file could be made more flexible; some of the register storage could be content-addressable, or part of it might be run as a stack or queue.

Memory interface : Modern microprocessors are already evolving towards having a very complex memory interface which can operate at peak efficiency with a large number of different external memory configurations. DPGA is already a contender for building these interfaces on microprocessor chips, and the opportunities can only increase as even more memory technologies become common.

DMA, device interfaces, and communications controllers : There is a massive opportunity for building specialised input/output processors. These could be configured to navigate directly through an application data structure in memory or could directly support some complex external communications protocol, such as ATM or ethernet.

Microcode store : The microcode store is becoming a large consumer of silicon in many modern microprocessors. The opportunity for DPGA technology is to store just the parts of microcode that are needed, and possibly to change the instruction set dynamically, as required by the current application. It is also possible to compress the size of the store by using PLA-style implementation where extensive opportunities exist for logic sharing and special-purpose microcode decoding logic.

Instruction decode, scheduling, and pipeline support : DPGA resources could be used to construct instruction decode operations which target a possibly changing instruction set, or which recognise instruction sequences in order to optimise or schedule the operations called for. The efficient running of the pipeline is increasingly important for high performance processors, and there are opportunities with DPGA technology to adapt the scheduling and optimisation strategies to the current application, or even to the current statistics of its operation.

Floating-point processor : The core of the FPP is a large and already a well-evolved unit with few opportunities for DPGA implementation unless non-standard formats are to be supported. Around the core of the FPU, there is opportunity for application-specific operations using the core. For instance FPU results could be extracted in both normalised and de-normalised form simultaneously or an operation such as root-mean-square might be made primitive.

Interrupt and misc. control : Just as the external bus of the microprocessor can be adapted to a wide range of environments by the use of DPGA, so can the miscellaneous control signals generated and consumed by the processor. Interrupt control is one clear example where application-specific behaviour might be beneficial

Novel components : It is possible to add novel special-purpose units at almost any point in the relatively fixed architecture of the processor. These might be best regarded as application-specific co-processors; a topic which has already been covered. However it is worth noting that there is the possibility to add many more than one co-processor and that some of them might be quite small.

The processor construction kit on a chip

If we take a child's construction kit with large numbers of a few simple building blocks as an analogy for today's DPGAs, then the development suggested here would be analogous to the production of specific models in the same child's construction kit framework, but with a host of specialised parts that support the construction of that particular model, or class of models only.

On the chip we would expect to see an extensive bus-based routing system in addition to a more traditional DPGA routing scheme so that processors with more than one internal bus could be constructed. We would expect to see multiple ALU structures, register files, and pipeline registers associated with the bus structures. A programmable external memory interface, with the additional possibility of a supporting cache, would be essential.

There are clearly some very difficult issues concerning what number and size of each of these resources to provide on the chip, and what portion of the chip should be devoted to traditional fine-grained DPGA resources. The evolutionary path is likely to be that these higher-level functions are added in a piecemeal fashion, probably in response to the demands of niche markets. Particularly in the early stages of development it will be necessary to maintain a high level of fine-grained resources before it becomes clear what mixes of fixed functionality blocks can become commodity items.

Conclusions

We have presented a number of ways in which the architecture of solutions to applications problems might change radically in the future. Some of the developments in this paper are speculative, others are happening already. We feel that there is no doubt but that reconfigurable hardware must be an increasingly important component in future systems and that it must inevitably be integrated with microprocessors, and finally change their architecture.

Reconfigurable hardware seems to have all the right characteristics to ensure its long-term position in the market. It is easy to design and fabricate; it can usefully consume the ever-increasing number of transistors provided for a given price by advances in VLSI technology; it reduces the design cost of new systems; it reduces the time to market for new systems; it allows systems to be upgraded up to the very last moment before they leave the factory, and also allows for post-delivery upgrades via floppy disc or telephone network.

With advanced tools such as we are trying to develop at Oxford, there will be a radical change in the skills base needed to implement tomorrow's high performance systems. It becomes feasible to implement complex hardware/software systems using mainly programmer effort, and in cases where an appropriate hardware platform with reconfigurable hardware resources already exists, it is feasible that the entire system can be produced by a software team alone.

Until arbitrary programs can be transformed into efficient hardware implementations, it will be necessary for the programmers to think carefully about the hardware programs they write. They will have to explicitly decide what things happen in parallel, perhaps to the extent of knowing what is being executed on each clock cycle. It is actually the case that these considerations are part of the everyday life of the electronics engineer, so it might well be that many of the programmers who develop such systems in the future will be electronic engineers moving to a more abstract design domain, rather than programmers moving to a more concrete one.

This is rather speculative paper and no doubt some of ideas contained in here will not see the commercial light of day. It is equally certain that many things will come to pass that have not been thought of here. However, it seems absolutely certain that there are a number of new ideas around, and hardware compilation and DPGAs are key among them, which are going to change we design and build digital systems. The author is certainly looking forward to the next decade of research, convinced that it is going to be even more exciting than the last few years has already proven to be.

Acknowledgements

This work which underlies this paper has been supported over the last few years by the European Union (Esprit/OMI programme), the U.K. Engineering and Physical Sciences Research Council (EPSRC), Advanced Risc Machines Ltd. (ARM), Sharp European Laboratories (SEL), Inmos Ltd. (SGS Thompson), Atmel Corp., Xilinx Development Corp., Music Semiconductor, and Oxford University Computing Laboratory, as well as others who wish to remain nameless. I particularly wish to thank Tony Stansfield for his work on the CAM-DPGA architecture while working for the Computing Laboratory, and I also wish to thank Geoffrey Randall for producing the diagrams in this paper and Jon Saul for his comments on the first draft.

References

[1] J. M. Arnold, D. A. Buell, and E. G. Davis. Splash 2. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 316–324, June 1992.

[2] P. M. Athanas and H. F. Silverman. Processor reconfiguration through instruction-set metamorphosis. *Computer*, 26(3):11–18, March 1993.

[3] P. Bertin, D. Roncin, and J. Vuillemin. Programmable active memories: a performance assessment. In G. Borriello and C. Ebeling, editors, *Research on Integrated Systems: Proceedings of the 1993 Symposium*, pages 88–102, 1993.

[4] J.P. Bowen, editor. *Towards Verified Systems.* Real-Time Safety Critical Systems Series. Elsevier, 1993. In preparation.

[5] D. A. Buell and K. L. Pocek, editors. *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1994.

[6] Steve Churcher, Tom Kean, and Bill Wilkie. Xc6200 fastmap processor interface. In Will Moore and Wayne Luk, editors, *Field Programmable Logic and Applications*, number 975 in Lecture Notes In Computer Science, pages 36 – 43. Springer, 1995. Procedings of the 5th International Workshop.

[7] Steve Guccione. List of fpga-based computing machines. world-wide web.

[8] Jifeng He, Ian Page, and Jonathan Bowen. Towards a provably correct hardware implementation of Occam. In G.J. Milne and L. Pierre, editors, *Correct Hardware Design and Verification Methods, Proc. IFIP WG10.2 Advanced Research Working Conference, CHARME'93*, volume 683 of *Lecture notes in Computer Science*, pages 214–225. Springer-Verlag, 1993.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall, 1985.

[10] C.A.R. Hoare and Ian Page. Hardware and software : The closing gap. *Transputer Communications*, 2(2):69–90, June 1994.

[11] Inmos. *Occam 2 Reference Manual.* International Series in Computer Science. Prentice-Hall, 1988.

[12] Inmos. *The Transputer Development and iq systems Databook.* Inmos Ltd., 1991.

[13] T. Kean and J. Gray. Configurable hardware: Two case studies of micro-grain computation. *Journal of VLSI Signal Processing*, 2(1):9–16, September 1990.

[14] Adrian Lawrence, Andrew Kay, Wayne Luk, Toshio Nomura, and Ian Page. Using reconfigurable hardware to speed up product development and performance. In W. Luk and W. Moore, editors, *FPL95*, Lecture Notes in Computer Science. Springer Verlag, 1995.

[15] Adrian Lawrence, Ian Page, Wayne Luk, Andrew Kay, and Toshio Nomura. Using reconfigurable hardware to speed up product development and performance. In *Proc. JFIT Conference.* JFIT, 1993.

[16] Will Moore and Wayne Luk, editors. *FPGAs.* Abingdon EE&CS books, 1991.

[17] Will Moore and Wayne Luk, editors. *More FPGAs.* Abingdon EE&CS books, 1994.

[18] I. Page. Reconfigurable processor architectures. *Microprocessors and Microsystems*, 1996.

[19] Ian Page. Automatic design and implementation of microprocessors. In *Proceedings of WoTUG-17*, pages 190–204, Amsterdam, April 1994. IOS Press. ISBN 90-5199-1630.

[20] Ian Page. Parametrised processor generation. In Will Moore and Wayne Luk, editors, *More FPGAs.* Abingdon EE&CS books, 1994.

[21] Ian Page. *Hardware Compilation Research Group World-Wide Web Pages.* Oxford University Computing Laboratory, 1995.

[22] Ian Page and Wayne Luk. Compiling occam into FPGAs. In Will Moore and Wayne Luk, editors, *FPGAs*, pages 271–283. Abingdon EE&CS books, 1991.

[23] Greg Snider, Philip Kuekes, W. Bruce Culbertson, Richard J. Carter, Arnold S. Berger, and Rick Amerson. The teramac configurable computer engine. In Will Moore and Wayne Luk, editors, *Field Programmable Logic and Applications*, number 975 in Lecture Notes In Computer Science, pages 44–53. Springer, 1995. Procedings of the 5th International Workshop.

[24] Michael Spivey and Ian Page. How to program in handel. Technical report, see http://www.comlab.ox.ac.uk/oucl/hwcomp.html, Oxford University Computing Laboratory, 1993.

[25] Anthony Stansfield and Ian Page. The design of a new FPGA architecture. In W. Luk and W. Moore, editors, *FPL95*, Lecture Notes in Computer Science. Springer Verlag, 1995.

[26] Sundance Multiprocessor Technology Ltd., 4 Market Square, Amersham, Bucks HP7 0DQ, U.K. *Product Overview*, 1995.

[27] Xilinx, San Jose, CA 95124. *The Programmable Gate Array Data Book (1993).*