Reconfigurable Processors

Ian Page,
Oxford University Hardware Compilation Group
Invited Keynote Address for Heathrow PLD Conference
April 1995.

1 Introduction

We believe that radical changes are set to affect many aspects of computer architecture. 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. Using them, hardware can be built in milliseconds under purely software control; they are early embodiments of truly general-purpose hardware.

We argue that conventional microprocessors are a poor match for any *particular* application, though they support a wide range of applications quite well. We suggest that the solution is to use additional silicon to provide hardware which can be configured to support *any* application. By combining a conventional processor with a DPGA on a single chip, commodity pricing is maintained and yet the same part can be targetted effectively across a wide range of applications.

2 The Dynamically Programmable Gate Array

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 us 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 [FCC94], [ML94]. Our own work in this area was first reported in [PL91] where a modular system based on a closely-coupled 32-bit microprocessor and a DPGA was described. 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 [Pag95]. 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 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.

3 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 multi-media application. For the duration of any interaction with a software package, the user wants a machine which will run that application both fast and cost-effectively. The 'general-purpose' nature of the underlying compute engine is of no direct relevance to the user. It 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.

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.
- Add more fixed-function units.
- Add variable-function units.

The favoured third strategy 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.

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.

4 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 interface bus, consisting of sections for the parallel transfer of data, addresses and control information. This interface is used to construct an external memory system. By mapping device registers into the address space of the processor, the same bus structure is 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. In the future, it will become of increasing concern to support both fast and partial reconfiguration, but there is no fundamental problem to be overcome in order to achieve this.

4.1 The HARP reconfigurable processor

An example of a system with a tightly-coupled microprocessor and DPGA is shown in Fig. 1. This is a block diagram of our HARP reconfigurable computing platform which consists of a 32-bit RISC microprocessor (a T805 transputer) with 4 Mbytes of DRAM, closely coupled with a Xilinx 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 [Sun95]. It is an industry-standard TRAM module (size 4) [Inm91] 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 complicates the exchange of data and metastability problems have to be anticipated and dealt with in the interface.

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 [Inm88] language, and the part destined for

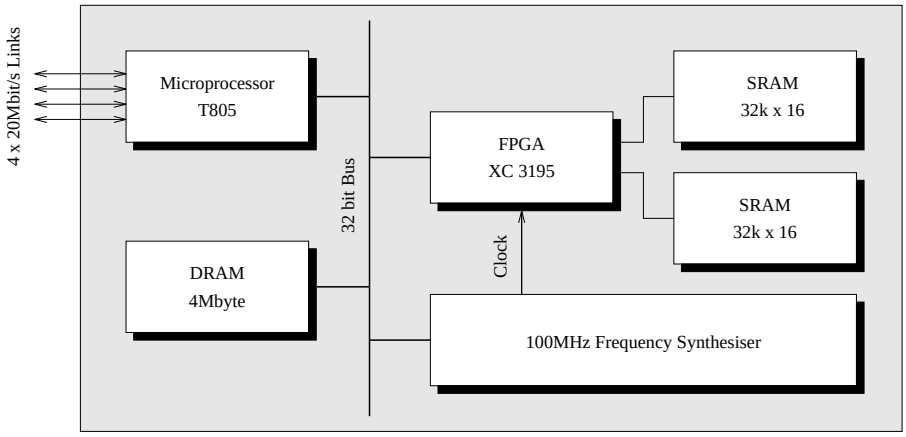

Figure 1: The HARP Reconfigurable Computer Module

hardware is written in Handel. 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 [Hoa85] 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 [HP94]. 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. 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.

4.2 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
- Remote procedure call
- Client-server model
- Parallel process

4.3 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 alone. Three different memory models suggest themselves:

- DPGA uses no external memory
- DPGA shares microprocessor memory
- DPGA has its own local memory

4.4 Models for DPGA Memory Operation

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

- Memory holds a complete data set
- Fifo queue
- Cache

4.5 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
- Application-specific microprocessor
- Sequential re-use of the DPGA
- Multiple simultaneous use
- On demand usage

4.6 Application-Specific Processor

The application-specific microprocessor 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.

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 [Pag94a], [Pag94b]. 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.

5 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.

5.1 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
- Memory interface
- Data transfer
- Block transfer
- DMA engine(s)
- On-chip cache
- Synchronisation
- Specialised Communication Interfaces

6 Combining Processor and DPGA on the Same Chip

If our predictions about the relevance of DPGA co-processors prove to be 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 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 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.

6.1 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
- Register file
- Memory interface
- DMA, device interfaces, and communications controllers
- Microcode store
- Instruction decode, scheduling, and pipeline support
- Floating-point processor
- Interrupt and misc. control
- Novel components

6.2 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 of 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.

Since it is already possible to design microprocessors automatically for a fine-grained target architecture, it should be relatively easy to adapt and improve these schemes for a higher level target architecture.

7 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 as 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 basis needed to implement new 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 five years has already proven to be.

8 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.

References

[FCC94] *FPGAs for Custom Computing Machines.* IEEE, 1994.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall, 1985.

[HP94] C.A.R. Hoare and Ian Page. Hardware and software : The closing gap. *Transputer Communications*, 2(2):69–90, June 1994.

[Inm88] Inmos. *Occam 2 Reference Manual.* International Series in Computer Science. Prentice-Hall, 1988.

[Inm91] Inmos. *The Transputer Development and iq systems Databook.* Inmos Ltd., 1991.

[ML94] Will Moore and Wayne Luk, editors. *More FPGAS.* Abingdon EE&CS books, 1994.

[Pag94a] 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.

[Pag94b] Ian Page. Parametrised processor generation. In Will Moore and Wayne Luk, editors, *More FPGAS.* Abingdon EE&CS books, 1994.

[Pag95] Ian Page. *Hardware Compilation Research Group World-Wide Web Pages.* Oxford University Computing Laboratory, 1995.

[PL91] 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.

[Sun95] Sundance Multiprocessor Technology Ltd., 4 Market Square, Amersham, Bucks HP7 0DQ, U.K. *Product Overview*, 1995.