The HARP Reconfigurable Computing System

Ian Page, Oxford University Hardware Compilation Group

October 1994.

1 Introduction

The Oxford Hardware Compilation Group is concerned with turning the process of hardware/software design and implementation into a largely programming activity. The group is committed to producing tools and techniques which are based in sound theory but which also work well in practice. For this reason we regard it as fundamental to produce working hardware-software co-systems, and that these should be of realistic, rather than toy, applications. This necessitates using hardware platforms which have one or more FPGA chips coupled with one or more microprocessors.

Field Programmable Gate Arrays are now sufficiently powerful that non-trivial algorithms can be mapped onto them with a resulting increase in speed of execution, for suitable algorithms, over conventional software implementations. The dynamic reconfigurability of some of these arrays means that one instance of the hardware can support many different algorithms during its lifetime. Particular applications in this technology will clearly not be as fast, or as cheap when massively replicated, as with custom ASICs. However, when using appropriate high-level design tools, the design cycle time can be enormously reduced and perhaps iterated towards the design goals many more times than otherwise possible. Also, dynamic reconfigurability allows a single application, or set of applications, to use the hardware in completely different ways at different times, thus ameliorating the higher parts cost of the FPGA chips.

To support our hardware/software co-design work, we have designed and built the HARP reconfigurable computer. This is a board-level system with the following major features:

- 32-bit RISC-style microprocessor (a T805 transputer),
- 4 Mbytes of dynamic RAM mainly for the transputer),
- Xilinx 3195 Field Programmable Gate Array,
- two independent banks of 32K x 16-bit fast static RAM (mainly for the FPGA),
- 100MHz frequency synthesiser for arbitrary FPGA clock generation,
- a parallel expansion port connected to the FPGA,
- industry-standard TRAM board (size 6 = 165 x 84mm),
- commercially available TRAM motherboards allow for easy integration into a variety of host systems, or for connecting multiple HARP boards together.

The FPGA is integrated with the microprocessor bus and can be completely reconfigured at any time by the processor. Once the FPGA is configured, and an appropriate clock frequency is established by programming the frequency synthesiser, the FPGA becomes a true co-processor. It has its own local memory and also has full access to the bus for high-speed communications to the microprocessor.

A typical application program can often be partitioned into two parts. A compute-intensive part is selected which might benefit significantly from a direct hardware implementation, while the other part may be so bulky and little used that it is best represented in the normal way as machine code for some suitable processor. The hardware compiler can be used alongside a conventional compiler so that flexible 'dual-paradigm' implementations can be produced by this partitioning process. Many different implementations of a program can be generated by merely altering the partitioning of the program and recompiling. Thus, software investment may be protected in the case that it is necessary to produce a number of different implementations of the same program, or similar programs, across a range of cost/performance characteristics.

The system is generally used for generating fast 'inner-loop' hardware, for instance in a real-time video motion tracker. The system has also been used to build, for example, a 16-bit microprocessor with an instruction set based on the Inmos transputer from what is essentially an instruction-set simulator for the processor, written in occam. We have also generated a number of application-specific microprocessors which have been designed completely automatically from an inspection of the programs they must execute.

Yet another style of application puts complex input/output device handlers into the FPGA with the majority of the application proper being supported by the microprocessor. It seems that FPGAs are particularly good for building the complex, and error-prone, interfaces to devices, ASICs and communications protocols. Having these interfaces described in a programming language can help to ensure clarity, correctness and flexibility in their implementation.

2 The Handel Language

Handel is a simple programming language designed for compiling programs into hardware implementations. Handel is not a hardware description language. Indeed it is our long-term aim to investigate how a single programming language can be used effectively for creating systems with both hardware and software components. We believe that it is only by using the paradigm of computer programming, together with its mathematical underpinnings, that we can hope to address tomorrow's large-scale design problems. This argument has even more force when we consider the design and implementation of parallel, safety-critical, or redundant systems.

The Handel language is based on the CSP algebra and its programming language counterpart, occam. The primitive resources available to a Handel program are variable-width integers, arrays of integers, and channels. The language supports expressions based on the usual arithmetic and logical operations, as well as conditionals, shared sub-expressions. and bit-field extraction and concatenation operators. Handel also supports both simple and simultaneous assignment, sequential and parallel composition, communication and al-

ternation, the usual While, If and Case control structures, as well as a restricted form of procedure calling.

3 FPGAs

Making it easier for programmers to produce circuits corresponding to their programs would be of little use unless there were some way of building these hardware circuits easily. Fortunately, there is a newly-available technology which does exactly this.

The Field Programmable Gate Array, or FPGA, is a relatively new form of integrated circuit which can be programmed to act as almost any digital circuit, subject to the physical limitations of chip size. The type of FPGA we use is a standard commercial chip consisting, essentially, of an array of gates and flip-flops that can be connected together via programmable switches, where the programmable elements are all implemented using static RAM. Changing the hardware configuration is accomplished by writing an appropriate set of bits to the RAM and can be repeated as often as required.

The major difference between this and previous hardware implementation technologies, is that the hardware can be reprogrammed rapidly by a purely software process. Today's FPGA chips can each implement a circuit with an equivalent complexity of some 20,000 gates in a matter of milliseconds. Even a single FPGA is capable of hosting a small application program kernel and the size and speed of FPGAs is increasingly rapidly as a result of progress in VLSI technology. Their use allows design decisions to be deferred, or implementations to be upgraded while in service. These advantages and others have already made FPGA sales the fastest-growing segment of the digital integrated circuit market.

However, many current uses of FPGAs ignore one of their most interesting characteristics. In a typical current application, one or more FPGA chips will be configured on system power-up and will remain near-permanently in that state. The ability to reconfigure FPGAs during system operation holds out the possibility that tomorrow's computing systems can use reprogrammable hardware as a flexible resource which can be deployed to support whatever computation is running at the time. Thus, reconfigurable hardware may bring benefits similar to those offered by 'reconfigurable software' (i.e. the stored-program computer).

In our HARP system, the FPGA subsystem is perhaps best regarded as a flexible co-processor. Once the microprocessor has loaded a configuration into the FPGA, they become equal partners in the computation. However, unlike a normal co-processor, the hardware in the FPGA can be changed completely at any instant. If this style of flexible hardware/software implementation becomes widely used, we can also expect to see a corresponding development of microprocessor architecture in which a substantial amount of reconfigurable hardware is embedded in the microprocessor core. Indeed we regard the whole HARP board itself as an experimental prototype of a future generation of microprocessors.

3.1 HARP Architecture

In order to explore hardware/software co-designs, we have designed and built the HARP board. This is a small, 17 x 9 cm, printed circuit board which conforms to the Inmos TRAM standard. It is a daughter-board which can very simply be integrated with commercially available host boards for many computers, and with a wide variety of other parallel computing and input/output modules. These modules can readily be joined together to make larger systems.

Our HARP module consists of two computing systems intimately-connected together; one is based on a T805 microprocessor, and the other a large Xilinx FPGA.

The T805 processor is convenient because it supports parallelism in general, and the occam language in particular. The HARP board equips the T805 with 4Mbytes of dynamic RAM. Its four 20Mbit/sec links are taken to the standard external TRAM ports thus providing the primary means of communication between the HARP board and its environment. The unused FPGA pins are also taken to a ribbon-cable connector so that high-bandwidth connections to external devices may be supported.

The T805 has control of the configuration port of the FPGA and can completely reconfigure it in a matter of milliseconds. A 100MHz frequency synthesiser is also under the control of the transputer and can be programmed to generate an arbitrary clock signal for the FPGA. The HARP frequency synthesiser is very stable and locks quickly and reliably. Once the FPGA has been configured, it operates as a true co-processor to the transputer; the major difference from a standard co-processor is of course that it can be made highly application specific simply by software reconfiguration.

The FPGA chip has access to almost every transputer bus signal so that high-speed communications can be established between the two processors. The FPGA has access to two independent banks of 32k x 16-bit fast SRAM. These can normally be accessed in a single cycle of the FPGA hardware. When necessary, a 'transparent' FPGA configuration can be loaded which maps the SRAMs directly onto the transputer bus. Since it has full bus access, the FPGA can also access the DRAM memory if a suitable DRAM driver circuit is loaded.

Since the FPGA can be dynamically reconfigured, then at one instant it might be supporting graphics or image-processing algorithms, and at the next it might be supporting data compression, spell-checking, or pattern matching.

4 Application : Video Motion Tracking

This application is described as an example of the use of hardware compilation in a challenging application arena. The task is to extract from a real-time video source a list of the bounding boxes of objects which are moving. The context of this application is an automatic security camera system which will usually relay a wide-angle view of the entire visual field, but which will pan and zoom the camera onto any moving objects within the scene. The goal is to record high quality, close-up video images of all the 'interesting' moving objects in a scene, rather than a single unvarying wide-angle view of the area under surveillance.

The application works by differencing successive digitised images and then thresholding the differenced image. An edge-following algorithm then looks for the over-threshold changes and traces out the (8-connected) boundaries of the changed regions. This clumps together the many pixel-level changes into a small number of changed regions which, at best, will correspond with individual moving objects in the scene. The bounding rectangles of the separate regions are then calculated, and these rectangle descriptors are then tracked by a Kalman filter. The filter attempts to follow their progress over time, even in the presence of noise, by both modelling and predicting their temporal behaviour in terms of their position and size, and the first derivatives of these quantities.

The frame differencing, edge following, and bounding rectangle computation are all performed in the FPGA on a single HARP board. This hardware was generated automatically from a Handel program. The Kalman filter, image stream transfer, and user interface are implemented in occam running on the HARP transputer. The performance is around 20 frames/sec on a 128 x 128 grayscale image, with 25 frames/sec being now within our sights. When this same application is run on the microprocessor alone, its performance drops by a factor of about six. To match this performance increase in software alone would obviously require the use of at least six similar microprocessors, and it is not at all clear how the computation could be partitioned to achieve this easily.

The current bottleneck in the process is the transputer's inability to move the images between the frame-grabber and the FPGA fast enough over its serial links - hence the limitation to 128 x 128 pixel images. This type of application really needs a hardware environment where the video images are directly accessible to the FPGA. We hope to design and build an advanced version of HARP with multi-media capability which will be an excellent host for this style of application. However, the performance we are achieving from a single HARP board is still very respectable in cost/performance terms.

Perhaps the most impressive feature of the application is that this fully operational, high-bandwidth combination of hardware and software was constructed by an undergraduate programmer, Matthew Bowen (thanks for a wonderful job, Matt!) who still proudly proclaims that he has no knowledge of hardware! I feel it underlines the realism of our goal to turn hardware/software co-design into just another smart compilation process.

5 Future Work

In coupling FPGA technology with hardware compilation, we are trying to provide environments where programmers, as well as hardware engineers, can build systems which take advantage of the greater speed offered by hardware. In a recent example, a programmer in our group having little understanding of hardware, has produced a small, elegant, spelling-checker in hardware that runs more than 25 times faster than a similar implementation of the same algorithm on a transputer. We will not pretend that the development of this, and other similar applications, was nearly as easy or straightforward as we would like, but we have demonstrated feasibility, and it gives us hope that our main goals ultimately will be achieved. It now seems reasonable to envisage that, in time, there may well be programmers who neither know nor care that the program they have just written and compiled is running as hardware, software, or some combination of the two.

One use of FPGA chips which we are beginning to investigate is the use of on-the-fly reconfiguration, for which we might use the term 'virtual hardware' by analogy with virtual memory. In this scenario, an application runs as far as it can with whatever hardware is loaded. When it can run no further, the environment swaps in further hardware, on demand, to enable processing to continue. The occam `SEQ` constructor provides a natural structure for algorithms that work in this way. We also expect our theoretical models and proof techniques to extend to such systems.

A particular area which is attracting our interest is that of multi-media applications. These are characterised by (i) a wide variety of information representations, many of them needing large amounts of data storage, with consequent emphasis on compression and efficient data transfer, (ii) computationally-intensive conversion programs which can extract high-level information from one domain of data and inject it into another, and (iii) interfaces with a wide variety of input/output devices. The reconfigurable FPGA can lend support in each of these areas. As one example, the accuracy and detail of graphics rendering algorithms embedded in FPGAs could be traded off against throughput if the system became overloaded by a request for a high-speed animation sequence. Similarly in a multimedia application, the same physical FPGA hardware could be supporting speech synthesis at one moment and video compression at the next. The occam `CASE` constructor provides a natural way of exploiting this style of reconfiguration.

We have looked at a number of applications of hardware compilation coupled with implementation via FPGA technology. Some of these have been speculative; others are a practical reality already in our laboratory and are being actively transferred into industrial practice. Our goal is to use these devices and our hardware/software compilation techniques to build flexible and dependable computing systems. We look forward to building a system in the future where users write programs or invoke applications, and literally have no idea whether the implementation generated is in software, in hardware, or in some mixture of the two; the compilation system will have ensured that the user's high-level constraints on cost and performance are met as closely as possible by the implementation.

References

[1] C A R Hoare *Communicating Sequential Processes.* 1985. Prentice-Hall International, UK, Ltd. ISBN 0–13–153271–5.

[2] *occam 2 Reference Manual.* 1988. Prentice-Hall International, UK, Ltd. ISBN 0-13-629312-3. Inmos Limited. Document 72 occ 45 01.

[3] Will R Moore and Wayne Luk, Eds. *FPGAs.* 1991. Abingdon EE&CS Books.

[4] Will R Moore and Wayne Luk, Eds. *More FPGAs.* 1993. Abingdon EE&CS Books.

[5] Page,I and Luk,W, *Compiling occam into FPGAs,* in [3].

[6] Luk,W and Page,I, *Parameterising Designs for FPGAs.* in [3].

[7] Page I, Luk W and Lau H *Hardware Compilation for FPGAs: Imperative and Declarative Approaches for a Robotics Interface.* 1993.

[8] Page,I *Automatic Design and Implementation of Microprocessors.* 1994 Transputer Communications. (To be published.)

[9] He,J and Page,I *A Normal Form Approach to Hardware Implementation of Occam.* 1992. Oxford University Computing Laboratory Technical Report.

[10] He J, Page I and Bowen J P. *Towards a provably correct hardware implementation of Occam.* 1993. In Correct Hardware Design and Verification Methods, Proc. IFIP WG10.2 Advanced Research Working Conference, CHARME'93. Edited by G.J. Milne and L. Pierre. Springer-Verlag. Lecture Notes in Computer Science, **683**,214–225,1993.

[11] Page,I *Parametrised Processor Generation.* 1993. in [4].

[12] Luk W, Ferguson D and Page I *Structured Compilation of Parallel Programs into Hardware.* 1993. in [4].

[13] A E Lawrence *HARP1 User Manual.*

[14] *Dual-in-Line Transputer Modules (TRAMs).* Inmos Technical Note 29 in, for example, *Transputer Development and iQ Systems Databook.* 2nd Edition, 1991.

[15] *The Programmable Logic Data Book.* 1993. PN 0401096-01. Xilinx,Ltd. Byfleet, Surrey KT14 7DU.

[16] *The Transputer Databook.* Third edition. 1992. Inmos. (SGS-THOMSON). Inmos document 72 TRN 203 02. Order code DBTRANST/3.