Customisable EPIC Processor: Architecture and Tools

W.W.S. Chu, R.G. Dimond, S. Perrott, S.P. Seng and W. Luk
Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, UK

Abstract

This paper describes a customisable architecture and the associated tools for a prototype EPIC (Explicitly Parallel Instruction Computing) processor. Possible customisations include varying the number of registers and functional units, which are specified at compile-time. This facilitates the exploration of performance/area trade-off for different EPIC implementations. We describe the tools for this EPIC processor, which include a compiler and an assembler based on the Trimaran framework. Various pipelined EPIC designs have been implemented using Field Programmable Gate Arrays (FPGAs); the one with 4 ALUs at 41.8 MHz can run a DCT application 5 times faster than the StrongARM SA-110 processor at 100 MHz.

1. Introduction

Rapid advances in reconfigurable hardware technology have enabled the implementation of instruction processors using Field Programmable Gate Arrays (FPGAs). Such implementations can often be customised to meet the requirements for a particular application. Examples of these reconfigurable instruction processors include Nios [1] and MicroBlaze [15], and various architectures based on microcontrollers, MIPS processors and Java virtual machines. There are also commercial offerings [11, 12] for rapid development of customisable instruction processors, targeting mainly implementations using ASIC (Application Specific Integrated Circuit) technology.

Customisable instruction processors offer the potential advantage of improved performance with reduced resource usage [9]. This is achieved by eliminating unused instructions, or by creating a new custom instruction to replace a group of frequently-used instructions. Such optimisations allow unnecessary hardware to be removed, and the overhead of instruction fetch and decode to be reduced [10].

Many contemporary instruction processors are superscalar: they execute multiple instructions concurrently by exploiting Instruction Level Parallelism (ILP). This is achieved by replicated functional units and sophisticated scheduling hardware to perform dependence analysis at run time.

The complexity of run-time dependence analysis, however, militates against implementation of customisable superscalar processors in FPGA technology. For effective handling of demanding applications, such as those involving real-time operations, an alternative architecture is desirable. Explicitly Parallel Instruction Computing (EPIC) architectures [8], together with the corresponding adaptive variations [5], appear to be promising candidates [3].

EPIC processors are also designed to exploit ILP. Like superscalar processors, they contain multiple independent units for processing instructions in parallel. However, they avoid run-time dependence analysis by performing such analysis at compile-time. By shifting the burden of scheduling from run-time to compile-time, the resultant hardware simplicity enables the implementation of EPIC processors in FPGA technology.

To make it scalable and extensible, the architecture of the customisable EPIC processor is designed in a parametric and modular way. Possible parameterisations include varying the number of registers and functional units, which are specified at compile-time. In addition, as a result of its modularity, inclusion or exclusion of a custom instruction only requires modifications of the concerned functional unit. Elements such as the ALU can be readily customised to cover particular application requirements.

Such customisable designs provide a platform for designers to explore performance/area trade-offs for a specific application using different implementations; they are particularly useful in system environments for reconfigurable computing. Moreover, instead of designing a separate processor for each application, a parameterised description allows the same design to be customised in various ways to meet application requirements. This ensures that the investment in the design is preserved.

The purpose of this paper is to describe the architecture of a customisable EPIC processor. We also describe the associated tools, which include a compiler and an assembler based on the Trimaran framework [13]. Moreover, the per-

formance of our EPIC architecture is empirically compared with the StrongARM SA-110 processor.

The rest of the paper is organised as follows. Section 2 introduces the background for an EPIC processor. Section 3 describes the customisable architecture of our processor, for which the compiler and assembler are presented in Section 4. Section 5 summarises performance and resource usage results. Finally, Section 6 covers concluding remarks.

2. Background

To speed up processor execution, researchers are working on two main architectural styles: superscalar and Explicitly Parallel Instruction Computing (EPIC).

Superscalar processors contain replicated functional units such that multiple instructions can be executed concurrently. In the meantime it still provides a sequential processor model to both programmers and compilers. Concurrent execution is achieved by exploiting Instruction Level Parallelism (ILP) via run-time dependence analysis.

EPIC processors, on the other hand, also have multiple functional units and aim for maximum exploitation of ILP. However, what differentiates EPIC and superscalar processors is the way that this parallelism is discovered: Superscalar processors use logic to schedule instructions at runtime, while EPIC processors rely on compilers that can statically expose parallelism at compile time. The EPIC approach allows the elimination of the scheduling logic and thus simplification of the processor organisation. To enable dependence analysis at compile-time, relevant information on the processor organisation is made available to EPIC compilers for scheduling purposes.

In order to make the processor's behaviour more predictable by the compiler, out of order executions are not supported in EPIC processors. Even though the programmer can still regard the underlying processor as a sequential machine with the abstraction provided by the compiler, the compiler itself can no longer remain ignorant of the processor organisation.

One of the most significant architectural innovations of EPIC is the inclusion of predicated instructions [8]. Predicated instructions transform control dependence to data dependence, so that the processor need not execute instructions such as branch instructions speculatively. Instead, while waiting for the result on whether to branch or not, multiple branches are executed simultaneously. These branch instructions are associated with appropriate predicate registers, which are one-bit flags recording branching conditions. Only those instructions associated with a predicate register showing a true condition will be committed; others will be discarded.

OPCODE	DEST1	DEST2	SRC1	SRC2	PRED
15 bits	6 bits	6 bits	16 bits	16 bits	5 bits

Figure 1. The instruction format

Other EPIC features include:

- Speculative loading, which involves retrieving data from memory before it is required.
- Data placement in memory hierarchy, which is controlled explicitly by the compiler.
- Simple run-time decisions, to reduce cycle time with a high level of ILP.

3. EPIC Processor Design

By varying the number of registers and functional units, our customisable EPIC architecture can easily be scaled to tackle the task in hand. Moreover, its parametric and modular design facilitates both inclusion of custom instructions and exclusion of unused instructions. Such a design provides a platform for designers to explore performance/area trade-offs for a particular application.

3.1. Instruction Set Architecture

The instruction set supported in our EPIC processor is a proper subset of operations specified in the HPL-PD architecture [4], a meta-architecture encompassing a space of machines, each of which has a different amount of ILP and a different instruction set architecture. This chosen subset of instructions focuses on integer operations, including multiplication and division, which can be implemented efficiently on FPGAs. In our design, a big-endian architecture is adopted.

The instruction consists of 6 fields, which is shown in Fig. 1. DEST1, DEST2 are indices to general purpose registers, SRC1 and SRC2 are either literals or indices to registers, and PRED specifies a predicate register.

Each individual instruction has a fixed width of 64 bits, regardless of its type. This regularity facilitates instruction fetching and decoding, which eliminates the need to look ahead to determine the exact length of the next instruction. Besides, the opcode has been designed to minimise the Hamming distance between two instructions of the same type.

3.2. Processor Organisation

To make the design more customisable, our EPIC architecture, captured in the Handel-C language [2], is designed in a modular way. The architecture contains four

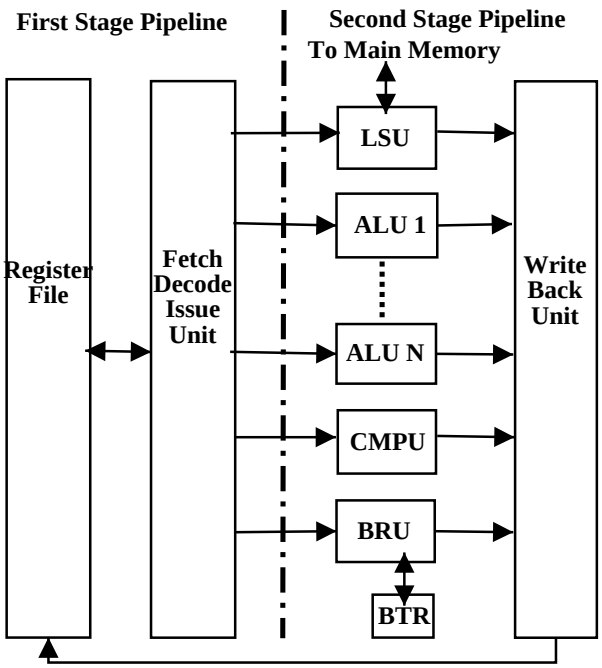

Figure 2. The datapath organisation

main types of elements: a collection of arithmetic and logic units (ALUs), a load/store unit (LSU), a comparison unit (CMPU), and a branch unit (BRU). These elements are sandwiched between a Fetch/Decode/Issue unit and a write back unit, which connect to a general purpose register file.

The organisation of the datapath is illustrated in Fig. 2. BTR stands for branch target register, which stores destination addresses which are calculated in advance and are likely to be required in the near future. The thick broken line in the middle signifies the division of the pipeline. Units of ALUs are denoted with a thin broken line as the exact number is customisable.

Currently, the prototype is a 2-stage pipeline implementation without cache. The Fetch/Decode/Issue unit constitutes the first stage of pipeline, with other units constituting the second.

As up to four instructions are issued per clock cycle, in this 2-stage pipeline design a maximum of eight reads from the Fetch/Decode/Issue unit and four writes from Write Back unit will come up simultaneously. However, being implemented with dual-port memory, which allows simultaneous read and write, the register file allows only two read/write operations in each cycle.

To tackle this challenge, a register file controller, which runs at quadruple the clock rate of the EPIC processor, is implemented. Therefore, any combinations of reads and writes amounting to eight operations are permitted in each processor cycle. Exceeding this limit would result in processor stall. Fortunately, this limitation is mitigated by forwarding of recently calculated results, which is also handled by the register file controller.

Regarding the access to memory, our design assumes that there are 4 external banks of memory, each 32-bits wide. Since each instruction is of 64 bits wide and a maximum of four instructions are issued per cycle, 256 bits must be read from the external memory in each cycle. To provide enough bandwidth, a memory controller which runs at twice the speed of the EPIC processor is implemented to oversee access to the main memory.

3.3. Processor Customisation

There are mainly two ways to customise the EPIC processor, by creation of customisable instructions or by the variation of its parameters.

Elements such as the ALU, the comparison unit and the load/store unit can be readily customised to cover particular application requirements. In most cases, only the concerned units require modification. For instance, ALUs do not need to support division if this operation is not required by the particular application program. This provides the basis for a scalable EPIC design, which can be customised to the application in hand.

In addition, our customisable EPIC description supports the following parameters:

- number of ALU units
- number of general purpose registers
- number of predicate registers
- number of branch target registers
- number of registers each instruction can use
- number of instructions per issue
- width of datapath and registers
- functionality of ALU

The main complication involving parameterisation is the pre-defined instruction format. The pre-defined instruction format assumes a range of possible values for some of the parameters above. For instance, as 6 bits are allocated to index a register, the maximum number of registers is assumed to be 64. Exceeding this limit requires a re-design of the instruction format.

For this reason, provision has been made for such adjustment, with the instruction width and the width of each individual field made parameterisable. The need for this flexibility is the reason for selecting an instruction width of 64 bits.

By default, the number of ALUs, general purpose registers, predicate registers and branch target registers are set to 4, 64, 32 and 16 respectively, with the width of datapath and registers being 32 bits long. Four instructions are issued in each cycle, provided that there is no resource conflict between them. All these parameters are instantiated in the configuration header file. Due to limited memory bandwidth, the number of instructions per issue is constrained between one and four.

4. Support for Application Development

4.1. Compiler

In the EPIC philosophy, the compiler not only statically schedules the order of instruction execution, but also controls the precise usage of all the hardware resources. It is therefore necessary for the compiler to possess detailed knowledge of the processor organisation, and be able to perform appropriate optimisations given the statistics of instruction usage.

Originally targeting the HPL-PD and ARM architectures, the Trimaran framework [13], which is a system for ILP research and contains modules for compilation and simulation, has been adapted to support our EPIC design.

Trimaran consists of three modules. In our current system, we employ the *IMPACT* module and the *elcor* module to support application compilation in our EPIC design.

Given an application program written in C, the *IMPACT* module is employed to perform machine independent optimisations. The *elcor* module will then statically schedule the instructions by performing dependence analysis and resource conflict avoidance.

Processor organisation information, including number of functional units, instruction issues per cycle and functionality of each module, is captured in the machine description language *HMDES* and serve as an input to *elcor*. By modifying the appropriate entries in the machine description file during customisation, the compiler is able to support our design, without the need for recompiling the compiler itself.

4.2. Assembler

To map the assembly code produced from Trimaran into EPIC machine code, an assembler, written in C++, is developed. To enable the assembler to adapt to EPIC processors with different customisations, the configuration header file is made available to the assembler.

While parsing the output from Trimaran, which is interspersed with instructions for its simulator module, the assembler filters the instructions for simulation purpose and counts the number of instructions actually available to execute in parallel. If necessary, no-op instructions are used to make up the difference.

For parameterisations, the assembler is able to support the design without the need for recompiling itself. Similarly, to create customisable instructions, there is no need for recompilation of the assembler itself. To adapt to such customisations, corresponding opcodes should be inserted into the configuration file.

5. Experimental Results

Experiments are performed to investigate the efficiency and the resource usage with varying number of ALUs. Xilinx Virtex II series devices, each containing up to 46592 configurable logic slices and up to 1.456 megabits of distributed configurable memory, are chosen as the target technology.

5.1. Resource Usage

Our design is implemented to evaluate its effectiveness on resource usage. The main results are summarised below:

- Currently, our prototype runs at 41.8MHz.
- Designs with 1, 2, 3 and 4 ALUs take up 4181, 6779, 9367 and 11931 slices respectively, in which each individual ALU occupies around 2600 slices.
- As the ALUs are arranged in parallel, varying the number of ALUs has little impact on the critical path; so is the case of enlarging the register file.
- The register file is mapped into SelectRam, the on-chip block RAM provided by Xilinx Virtex II series devices.
- As long as additional block RAM is available, increasing the size of register file has negligible effects on number of slices taken up.
- Multiplication is supported by on-chip block multiplier.

5.2. Comparison with the StrongARM SA-110

To evaluate the performance of the prototype, the performance of our EPIC designs equipped with one to four ALUs is measured against the StrongARM SA-110 processor. It is measured in terms of number of clock cycles consumed for executing a particular benchmark.

The number of cycles taken by our EPIC design is measured by the *ReaCT-ILP* module, a cycle-level simulator, from the Trimaran framework. The number of cycles for the StrongARM SA-110 processor is obtained by the ARM

Table 1. Summary of the number of clock cycles required for different benchmarks

	SHA	AES	DCT	Dijkstra
SA-110	15934277	1505572	140679660	97272080
1 ALU	14252100	8809870	29445100	58877500
2 ALUs	7486250	8513740	22191700	57257400
3 ALUs	5276340	8515610	20647500	57230000
4 ALUs	4172960	8514620	11426500	57219900

simulation program SimIt-ARM [7]. The results are summarised in Table 1.

The operation of these benchmarks is explained below:

- The SHA benchmark calculates the SHA-256 secure hash of a 256 by 256 image in the PPM format.
- The AES benchmark encrypts 'Hello AES World!' 1000 times and then decrypts it.
- The DCT benchmark does fixed-point Discrete Cosine Transform (DCT) encoding and decoding of a 256 by 256 image in the PPM format.
- The Dijkstra benchmark finds the shortest path between every pair of nodes in a large graph represented by an adjacency matrix using Dijkstra's algorithm.

From Table 1, we notice that in most cases, our EPIC designs manage to complete with fewer cycles than the SA-110. For instance, were the two processors run at the same clock speed, our EPIC design with 4 ALUs would be 1.7 times, 3.8 times, and 12.3 times faster than the SA-110 in the benchmarks Dijkstra, SHA and DCT respectively.

Comparing the number of clock cycles is useful when two processors have the same cycle time. If not, the variation in cycle time should be taken into account. Comparisons in terms of time to execute the benchmarks SHA, DCT and Dijkstra are shown in Fig. 3, 4 and 5 respectively, given that the SA-100 runs at 100MHz while our current EPIC processor runs at 41.8MHz.

These figures show not only the relative performance between the SA-110 and EPIC designs, but also how the performance varies as the number of ALUs increases. Execution time is calculated as a product of clock length and the number of clock cycles taken.

From these experiments, it is observed that EPIC processors equipped with several ALUs are able to outperform the SA-110 in the benchmarks SHA and DCT. For instance, running SHA the SA-110 takes 0.1593 seconds, while EPIC with 4 ALUs takes 0.09981 seconds, which is 60% faster. As for DCT, the EPIC processor with 4 ALUs is 515% faster than the SA-110. We can observe that in these arithmetic-intensive applications, the performance of EPIC increases

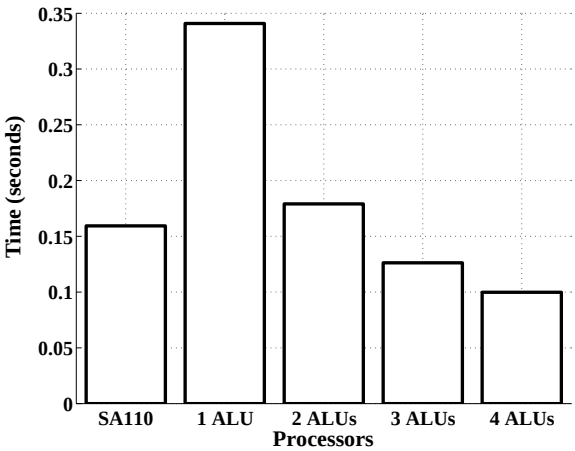

Figure 3. Execution time for SHA

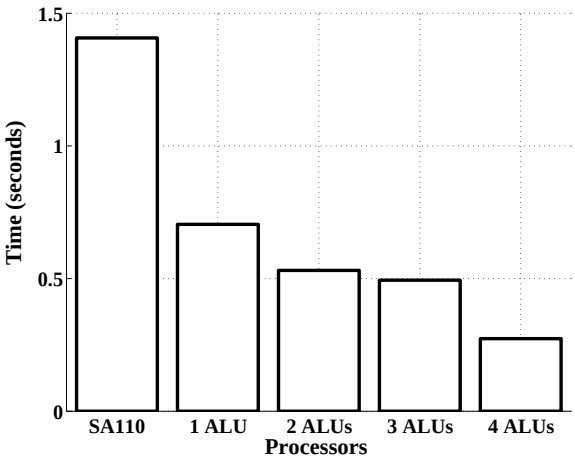

Figure 4. Execution time for DCT

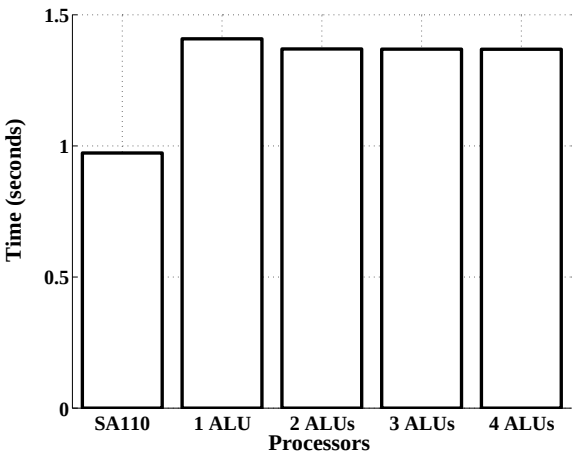

Figure 5. Execution time for Dijkstra

as the number of ALUs increases. On the other hand, for AES and Dijkstra, which involve relatively few arithmetic operations, the performance remains more or less the same regardless of the number of ALUs deployed.

These results show that the performance of our softcore EPIC processor is comparable to a hardcore processor. Despite the fact that the SA-110 is able to outperform our design in the benchmarks AES and Dijkstra, the results look encouraging. While the SA-110 is an established commercial product, the customisable EPIC processors developed in our research are not highly optimised. With further optimisations in the design of the datapath, a speedup in clock rate should be possible.

Furthermore, the advantage of customisability is demonstrated: depending on the processor's intended application, redundant hardware can be eradicated and additional hardware can be supplemented to optimise the performance/area trade-offs.

6. Concluding Remarks

Our customisable EPIC processor provides a prototype architecture which can be customised to a particular application. Because of its hardware simplicity, EPIC processors are well-suited to FPGA implementation.

Moreover, empirical results suggest that with suitable customisations, the performance of our design is comparable to the performance of a hardcore processor. This demonstrates the feasibility of implementing EPIC-style processor for integer computations on programmable logic. As we are still in the initial phase of research, with further optimisations in the datapath additional speedup should be possible.

Current and future work includes parameterising the level of pipelining, supporting automatic generation of custom instructions, and customising compiler optimisations for adaptive EPIC architectures [6]. We are also interested in characterising the trade-offs in performance, size and power consumption [14] of our customised EPIC processors.

Acknowledgements

The support of Celoxica Limited, Xilinx Inc. and the U.K. Engineering and Physical Sciences Research Council (Grant number GR/N 66599 and GR/R 31409) is gratefully acknowledged.

References

[1] Altera Corporation, *Nios Embedded Processor 32-Bit Programmer's Reference Manual*, http://www.altera.com.

[2] Celoxica Limited, *Handel-C Language Overview*, http://www.celoxica.com.

[3] V.S. Gheorghita, W.F. Wong, T. Mitra and S. Talla, A Co-simulation Study of Adaptive EPIC Computing, *Proc. IEEE International Conference on Field-Programmable Technology*, 2002, pp. 268–275.

[4] V. Kathail, M.S. Schlansker and B.R. Rau, *HPL-PD Architecture Specification: Version 1.1*, Technical Report HPL-93-80 (R.1), HP Laboratories, February 2000.

[5] K.V. Palem and S. Talla, Adaptive Explicitly Parallel Instruction Computing, *Proc. 4th Australasian Computer Architecture Conference*, January 1999.

[6] K.V. Palem, S. Talla and W.F. Wong, Compiler Optimizations for Adaptive EPIC Processors, *Proc. First International Workshop on Embedded Software*, LNCS 2211, Springer-Verlag, 2001, pp. 257–273.

[7] W. Qin and S. Malik, Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation, *Proc. 2003 Design Automation and Test in Europe Conference (DATE 03)*, pp. 556–561, 2003.

[8] M.S. Schlansker and B.R. Rau, EPIC: Explicitly Parallel Instruction Computing, *Computer*, February 2000, pp. 37–45.

[9] S.P. Seng, W. Luk and P.Y.K. Cheung, Customising Flexible Instruction Processors: A Tutorial Introduction, *Proc. Second International Workshop on Systems, Architectures, Modeling, and Simulation*, 2002.

[10] S.P. Seng, W. Luk and P.Y.K. Cheung, Run-time Adaptive Flexible Instruction Processors, *Proc. Field Programmable Logic and Applications*, LNCS 2438, Springer-Verlag, 2002, pp. 545–555.

[11] Target Compiler Technologies, http://www.retarget.com.

[12] Tensilica Incorporated, *Xtensa Architecture White Paper*, http://www.tensilica.com.

[13] Trimaran: An Infrastructure for Research in Instruction Level Parallelism, http://www.trimaran.org.

[14] F. Vermeulen, F. Catthoor, L. Nachtergaele, D. Verkest and H. De Man, Power-Efficient Flexible Processor Architecture for Embedded Applications, *IEEE Trans. on VLSI Systems*, Vol. 11, No. 3, 2003, pp. 376–385.

[15] Xilinx Inc., *MicroBlaze Processor Reference Guide*, http://www.xilinx.com.