Initial Analysis of the Proteus Architecture

Michael Dales

Department of Computing Science, University of Glasgow,
17 Lilybank Gardens, Glasgow, G12 8RZ, Scotland.
michael@dcs.gla.ac.uk

Abstract. The Proteus Architecture proposed a general purpose microprocessor with reconfigurable function units. The ProteanARM represents an ARM-based realisation of this concept. This paper describes the initial details of the ProteanARM architecture and demonstrates some performance benefits gained through the use of custom function units. Our examples show a promising performance increase compared to a standard ARM processor, with reconfiguration costs being quickly amortised.

1 Introduction

Field Programmable Logic (FPL) is an attractive solution to numerous problems, combining the flexibility of software with the performance of hardware. However, FPL has failed to enter the modern microcomputer arena, computation on desktop machines and workstations being the domain of either software or application specific expansion cards (e.g., graphics accelerators).

In an attempt to bring the benefits of FPL to the desktop machine we have proposed placing the FPL in the heart of the machine: the microprocessor itself. Our architecture, called the Proteus Architecture [2], has a Reconfigurable ALU (RALU) consisting of multiple Reconfigurable Function Units (RFUs) in the core of the processor. It allows applications to load custom instructions at run time. In this paper we describe an initial ARM-based implementation of the Proteus Architecture and examine its performance using a software simulator.

1.1 Motivation

Whilst combining FPL with microprocessors is certainly possible (see Section 1.2 for examples), building a processor that needs to work inside a desktop computer, with the workloads that entails, poses some interesting design challenges. A desktop processor that includes FPL must be designed with such use in mind and it must provide a rich enough set of mechanisms that the Operating System (OS) can efficiently manage the FPL resource, allowing many applications to make use of it.

The Protean Architecture takes into consideration OS issues such as resource contention, virtualisation, and overheads of state management. For example, having a single large array of FPL may lead to several applications trying to gain access to it at once, giving poor performance. Large amounts of state could lead to high context switch overheads. Our approach is to conduct a top–down design that addresses these issues, including appropriate architectural support.

1.2 Related Work

Here we only describe work directly relevant to the sections that follow. A wider discussion of related work can be found in [2]. The PRISC [6] architecture proposed having a Reduced Instruction Set Computing (RISC) processor augmented with multiple reconfigurable function units, though the authors only produced results for an implementation with a single reconfigurable unit. These function units are stateless, which avoids additional context switch overheads.

GARP [3] takes a MIPS core and extends it with a single large FPL array. A single large array is a point of resource contention, and so the authors have allowed configuration caching and have only minimal state to reduce the context switching overheads. The CoMPARE [7] processor links a traditional ALU and a Configurable Array Unit (CAU) that can be wired to operate individually, in parallel, or sequentially. To reduce reconfiguration times, the CAU supports partial reconfiguration, allowing only the necessary parts of the CAU to change, which is useful for keeping down circuit switching overheads. Like PRISC, the CoMPARE architecture has no state in its array.

Triscend produce a commercial combination of an ARM core and a Configurable Logic Unit (CLU) called the Triscend A7 [10]. The A7 is aimed at the Configurable System–on–Chip (CSoC) market; the CLU can be programmed only at start–up, and the processor core and CLU are individual entities. In the same vein there is the FIPSOC (Field Programmable System–on–Chip) from SIDSA [9]. The FIPSOC has a 8051 microcontroller core and a reconfigurable array on a single chip, but again both as distinct entities. The Infineon Carmel 20xx architecture [4] offers tighter integration. The 20xx is a Digital Signal Processing (DSP) processor with four configurable Execution Units (EUs), similar to the RFU of the Proteus Architecture. Infineon claim this gives the 20xx twice the performance of its predecessor. Like the A7, the 20xx cannot be reconfigured at run time.

2 ProteanARM Architecture

The general ideas behind the Proteus Architecture were initially outlined in [2]. In order to realise the Proteus Architecture an existing processor architecture was taken and modified. Although the ideas could be applied to high–power processors, such as the Alpha, a simple architecture suffices to demonstrate the initial concepts and enables faster implementation and testing. For this reason the popular ARM processor was chosen. The ARM has a simple pipelined core and, through the use of internal coprocessors, has a well–defined way to expand the instruction set. We based our model on an ARMv4 core [1]. This platform is designed to be a starting point for our work, and will change as the project progresses and experimental work provides us with a better set of requirements for the archtiecture and highlights problem areas.

The ProteanARM extends the traditional ARM datapath with an RALU connected to a second register file. These are joined to the datapath using the standard internal coprocessor model. The Protean coprocessor cannot generate addresses in this model, so it must utilise the main core when loading and storing data, but data operations execute completely separately and may potentially run in parallel.

The RALU consists of a group of RFUs, meaning that multiple custom instructions can be present on the processor at the same time. We have chosen to have eight RFUs, which would allow several applications to have multiple instructions loaded at once. Currently the fabric for the RFUs is based on the Configurable Logic Blocks (CLBs) used in the Xilinx Virtex device [11]. Based on the die size of a Xilinx XCV1000, and subtracting area for the ARM core, we estimate having 6000 CLBs to divide between the RFUs, giving 750 CLBs per RFU. Each RFU will take in two 32 bit words and return a single 32 bit word result. The register file in the Protean coprocessor has sixteen 32 bit registers. The internal coprocessors access main memory through the same interface as the main processor core, and so gain all the benefits of the caches present.

The operation of the Protean coprocessor is quite simple. There are four types of instruction, all of which fit into the coprocessor instruction space. These use existing instruction mnemonics, allowing current compilers to work without modification. The first three types of instruction cover the usual operations of moving a value between the Protean coprocessor's register file and memory, moving a value between the main register file and the Protean coprocessor's register file, and executing an instruction loaded in an RFU. The only new type of instruction is that used to load a bitstream into an RFU. This is currently done as a long atomic operation to simplify the initial processor design.

Custom instructions can potentially last many cycles. An instruction is fed a start signal by the processor on its first cycle. When the instruction is finished it will send a completion signal to the processor, which will then store the result in the destination register. State in RFUs is not reset between calls, allowing for pipelined instructions.

The clock speed for the ProteanARM depends upon the speed that the ARM core can run at and the speed at which we expect instructions in RFUs to run. To guess the performance of the ARM core produced using the same technology as the RFU fabric would require detailed knowledge of how the ARM processor is implemented, which we do not have available. But, we can safely say that it should run at least as fast as the ARM core on a lesser technology. Thus, as a starting point, we have assumed that we can run the ProteanARM at the same speed as an ARM720T, which is 40 MHz.

3 Initial Experiments

Using a software simulation of the ProteanARM we have obtained some initial data to assess the performance of such a processor. In this section we describe two experiments that compare the performance of the ProteanARM with that of a conventional ARM.

3.1 Audio Processing

The first example is an echo filter algorithm taken from an Intel MMX technical note [5]. The aim is to show that the ProteanARM can handle MMX type instructions in its RFUs. But, thanks to the flexibility of reconfigurable logic, we are also able to perform optimisations not possible in MMX.

The algorithm processes a WAVE–format audio file, which uses 8 bits per sample at an 8 kHz sampling rate. The samples are stored as unsigned values, but must be

normalised to be between -128 and 127 for processing. This is done by xoring each sample with 80_{16}. To add e echos of delay d to sample s we look back in the file's history and add attenuated values. The attenuation factor, G, is greater the further back in time we go. This is expressed as follows:

$$s'[n] = s[n] + \sum_{1 \leq i \leq e} (G^i \times s[n - i \times d])$$

The aim of the implementation is to use Single Instruction/Multiple Data (SIMD) techniques to work on four octets per instruction. The key operations of the MMX example are a SIMD signed add, a variable SIMD arithmetic shift left, and an xor operation to normalise the samples. We improve on this by building the normalisation into the add and shift instructions, by partially evaluating the xor with constant operation into the circuits' inputs and outputs where necessary. This level of customisation was only possible with reconfigurable logic.

3.2 Alpha Blending

Alpha blending is the process of superimposing images that contain a level of transparency (their alpha value). Images are made up using 32 bits per pixel, an octet for each of the Red, Green, Blue, and Alpha channels (referred to as RGBA format). Without the alpha channel, summing images fits nicely into the SIMD add with saturation instructions found in MMX. But summing RGBA is more complex. The equation for summing the alpha channel is different to that for summing the colour channels, and summing the colour channels depends on the result of summing the alpha channel. The equations for the alpha and colour addition can be seen below, with f and b referring to the front and back images:

$$A_n = \frac{A_f \times A_b}{255} \qquad C_n = \frac{((255 - A_f) \times C_f) + (\frac{A_f \times C_b \times (255 - A_b)}{255})}{255 - A_n}$$

The aim of this example is to experiment with a circuit that adds two pixels in a single instruction, fitting nicely into the 32 bit datapath. The alpha blending circuit is both larger and more complicated than the circuits used in the previous example. It implements numerous Booth's multipliers and naive dividers, requiring the circuit to include state and take multiple cycles to complete.

3.3 Results and Discussion

For the echo example, the shift instruction required 75 CLBs and the adder instruction 20 CLBs. Both take less than a single cycle to complete at 40 MHz. Using the new instructions reduces processing a single sample from 570 cycles (in optimised C code) to 98 cycles. The alpha blending instruction uses 436 CLBs and takes 27 cycles to generate a result, substantially less than the corresponding 377 cycles required by software.

The test applications were run on a plain ARM simulator and a ProteanARM simulator. The echo example was run with a 2 second sample and alpha blending on two images of 350x194 pixels (the size of a dialog box). The cycle counts can be seen in Table 1.

Application	Conventional ARM	ProteanARM
Echo DSP	1,643,550	319,903
Alpha blending	28,409,137	3,180,453

Table 1: Cycle count results from experiments

Both examples demonstrate a significant reduction in cycle counts, despite the overhead of reconfiguration. The echo example amortised the reconfiguration costs in processing 35 samples, and the alpha blending example after 124 pixels. Both thresholds are likely to be easily achieved in most practical examples. Despite the costs of reconfiguration being easily amortised, the experiments have highlighted that atomic loads of circuits (notably the alpha blending circuit, which was 50k) is unreasonable.

4 Conclusion

This paper has demonstrated the validity of the Proteus Architecture as a platform for further work in the field of hybrid FPL processors. The experiments showed a favourable performance for our base architecture. In other work not reported here, we have achieved positive results with the Twofish encryption algorithm [8] and are currently working on a data retrieval example.

This work was in part supported by SHEFC RDG Project 85, Design Cluster for System Level Integration.

References

[1] ARM Ltd. *ARM Architecture Reference Manual*, DDI 0100D edition, 2000.

[2] Michael Dales. The Proteus Processor — A Conventional CPU with Reconfigurable Functionality. In *9th International Workshop on Field Programmable Logic and Applications*, pages 431–437. Springer–Verlag, September 1999.

[3] John R. Hauser and John Wawrzynek. GARP: A MIPS processor with a reconfigurable coprocessor. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 12–21, Napa, CA, April 1997.

[4] Infineon. Infineon introduces configurable CARMEL DSP Core for 3G wireless and broadband communication applications. Infineon Press Release, March 2000.

[5] Intel. Using MMX Instructions to Implement Audio Echo Sound Effects. Unpublished Technical Report, 2001.

[6] Rahul Razdan and Michael D. Smith. High–Performance Microarchitectures with Hardware–Programmable Functional Units. In *Proc. 27th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 172–180, November 1994.

[7] S. Sawitzki, A. Gratz, and R. G. Spallek. CoMPARE: A Simple Reconfigurable Processor Architecture Exploiting Instruction Level Parallelism. In *Proceedings of the 5th Australasian Conference on Parallel and Real-Time Systems*, 1998.

[8] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. *Twofish: A 128–Bit Block Cipher*, June 1998. AES Proposal.

[9] Sidsa. *FIPSOC Mixed Signal System–on–Chip*. SIDSA Semiconductor Design Solutions, 2000.

[10] Triscend. *Triscend A7 Configurable System–on–Chip Family*. Triscend Corporation, 2000.

[11] Xilinx. *The Programmable Logic Data Book 1999*. Xilinx, 1999.