The Proteus Processor

A Conventional CPU with Reconfigurable Functionality

This poster describes the starting position for research beginning at the Department of Computing Science, University of Glasgow. The research will investigate a novel microprocessor design incorporating reconfigurable logic in its ALU, allowing the processor's function units to be customised to suit the currently running application. We argue that this architecture will provide a performance gain for a wide range of applications without additional programmer effort.

University of Glasgow

SRG Systems Research Group

XILINX®

by Michael Dales

URL: http://www.dcs.gla.ac.uk/~michael/phd/
email: michael@dcs.gla.ac.uk

Trends

Trend 1: CPU Technology

Silicon processes moving to smaller die sizes:
⇒ more gates available on ICs (including CPUs).
⇒ more application specific Function Units appearing in new CPUs (e.g. MMX instructions in Pentiums, 3D Now! in AMD processors).

Drawbacks: New function units are application specific:
⇒ lower utilisation of silicon than before. New function units set in sand.
⇒ may not exactly map onto programmer's problem or may become out of date (e.g. MMX works on 8 bpp whereas modern apps use 16 or 24 bpp).

Trend 2: FPGA and CPU research

People are researching ways to link CPUs and FPGAs. Current methods involve:
- CPU connected to FPGA over bus
- Placing a CPU framework on top of an FPGA
- Placing an FPGA inside a CPU framework

Drawbacks: None of the attempts so far are likely to replace my Pentium. Either:
- Good for certain types of application ⇒ not general enough.
- Radical new architecture ⇒ Problems with legacy software and training programmers

The most promising solution seems to be the FPGA inside a CPU framework.

Extrapolation from trends

- A good way of consuming the extra gates on a CPU is to add more function units...
- ...but new static function units tend to decay and be under utilised.

- CPU and FPGA linkage is a good thing...
- ...but currently suffers from a problem of abstracting the FPGA resource without breaking the traditional programming model.

From this we get...

IDEA!

Utilise the perfectly well understood CPU abstraction of the Function Unit to encapsulate the Field Programmable Logic (FPL) resource.

By placing the FPL inside the ALU of a conventional CPU we provide fine grain support for utilising reconfigurable logic in the heart of the computer. The FPL will be easily accessible from a program, and without breaking any of the conventional functionality expected of a processor.

Advantages:
- Uses existing CPU model - little learning overhead for the programmer.
- Applications can load custom Function Units that suit the specific problem they need to solve.
- Inherently takes advantage of existing CPU technologies like pipelining, superscalar functions, etc.
- Function Units can be upgraded to take advantage of progressions in that field (e.g. better/faster encryption routines in an encryption unit).

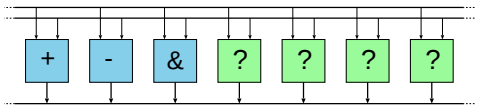
Under such a system, any application can load specific Function Units in order to get a speed increase. Possible applications include:
- DSP
- Multimedia
- Cryptography
- 3D graphics
- ...

It is not even necessary to have conceptually distinct function units - if an applications source is examined, then new FUs can be generated to replace any sequence of instructions.

Architecture Overview

The ALU of a Proteus Processor contains both static and Reconfigurable Function Units (RFUs) - making it a **Reconfigurable ALU (RALU)**. Both types of function unit are accessed in the traditional manner of the CPU - two operands and a single result. Notionally all the Function Units could be reconfigurable, however a large number of functions (such as integer math and boolean logic operations) are so common it makes sense to keep them.

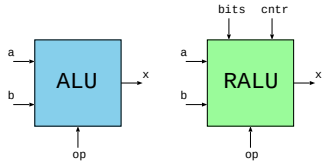
The inside of an RALU may look a little like this:



There are two obvious restrictions:
- A RFU is not as large as an FPGA (given that they share an IC with the CPU)
- The two inputs/one output set up

Whilst this may seem limiting, the abstraction makes the programmable logic much easier to access and utilise. Once a circuit is loaded into a RFU it can be invoked just like any other instruction.

A good way of seeing this abstraction is if we step back a level and treat the RALU as a black box and contrast it to a traditional ALU:



The diagram shows that both the traditional ALU and out RALU:
- Accept two operands, *a* and *b*
- Product a single result *x*
- Have the operation selected by the control lines *op*

In addition, the RALU has extra lines for reconfiguring the RALU: *bits* for receiving the circuit bitstream, and *ctrl* for the extra control lines that reconfiguration requires.

From this you can see that, other than for reconfiguring the RALU, its operational interface is the same as that of a conventional ALU.

In order to put everything into context, examine the diagram to the right. This is indicative of how a processor would include a RALU (in this case an ARM6). To make the change clearer, the traditional ALU has been left and the RFUs places in a separate RALU.

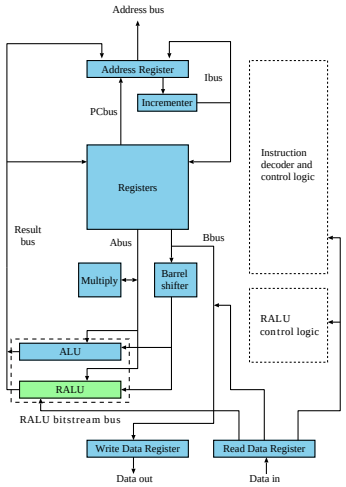
From this diagram it is possible to see how little of the processor has been changed. Only one extra bus has been added (the bitstream bus), along with a bunch of extra control logic for managing the RALU. Other than this the diagram is unchanged from a traditional ARM6 layout.

Under this model, existing ARM code would run on this architecture without modification - programs are not forced to use the RALU if they do not require to.

The remaining issue is how do programs set up the RALU for use and then utilise these new instructions? The sequence of events would run something like:
- Load circuit bitstream into memory
- Call an assembly language instruction to load an RFU with a bitstream from a given memory address
- Invoke new function using an RFU assembly call with what looks like any other instruction

Whether the program explicitly references a RFU (e.g. *"Load circuit into RFU 3"*) or the RFU is allocated into a free slot (e.g. *"Load circuit into next free RFU"*) is an area of research (see below for more details).

Below is what a piece of ARM code for explicitly using RFUs might look like:




```
...
lrfu 3, CIR_ADDR  ; load RFU 3 from addr
ldr r0, [r1]      ; Load the mask
orr r0, r0, r1    ; or with our mask
rfu3 r0, r2, r3   ; Invoke loaded circuit
str r0, [r1]      ; Put the end result away
...
```

Thus we have an architecture that:
- Supports the well understood abstraction of Function Units
- Allows applications easy access to Field Programmable Logic
- Does not require extensive modification to CPU design

Research Areas

Changing the processor at a fundamentally low level has repercussions throughout the system, touching many fields in computing science, of which only a subset are initially going to be tackled:

Hardware Design

- Relationship between the CPU and RALU:
  Should the we have a separate ALU and RALU? Should the RALU be superscalar?
- Type of FPL inside each RFU:
  Copy cell design from existing FPGA? Need new type of FPL?
- Size and number of RFUs:
  What size of circuit is practical for a single instruction? How many will a program use?
- Control logic to support RALU:
  Modifications to load the circuits and manage them.

Operating Systems

- RFU management:
  Similar problem to virtual memory management. Virtualised instruction sets:
  - "Paging" of instructions
  - Multiprogramming support
  - Virtual/Physical instruction mapping
- Security issues:
  Circuits could break OS security - cf. extendable OSs.

Programmer Support

- How are the circuits for RFUs generated:
  Libraries of circuits and code stubs?
  Circuits generated by compiler?
- Proving functional equivalence between code and circuits.

Research on this project will begin in October 1999, funded by EPSRC and Xilinx Edinburgh. In the initial phase we will address the low-level design issues of a microprocessor with a RALU. The viability of such a platform will be examined, investigating such issues as size and number of RFUs, and the type of logic that should go inside them. If this stage is successful then we plan to take an existing conventional CPU design and modify it to include a RALU. This will be simulated, and possibly prototyped, to prove the concept works.

This fundamental low-level work can then be used as a starting point for further work. This includes full analysis of the performance of the proposed architecture compared with both conventional microprocessors and the approaches taken by others, along with research into the topics discussed above, such as compiler support.