

UC Berkeley Computer Science

Subject: FIFO Register File
Date: June 30, 2006
From: Ivan Sutherland
UCIES #2006-is23

References:

UCIES# 2005-is02: FLEET – A One-Instruction Computer, Ivan Sutherland, 24 August 2005
UCIES# 2005-is03: Defining Some SHIPs, Ivan Sutherland, 24 August 2005

INTRODUCTION

This memo offers some ideas about a FIFO register file for FLEET. I'm writing this memo for the usual two reasons: to get my thoughts in order, and so others can criticize the ideas.

REGISTER FILE DESCRIPTION

Let us think of a FIFO register file of 16 integer FIFOs, each of which has eight entries. That's 128 integers of storage addressed with only four bits. I'm not fixed on that particular size, but I'd like to use it as a starting place for discussion. Specificity seems more important than accuracy, so I offer specific sizes.

I picked 16 FIFO registers quite arbitrarily. I picked eight entries for each FIFO because the delay in an eight stage FIFO is about the same as the pipeline delay of the switch fabric. A longer FIFO might serve as a throughput bottleneck, and a shorter FIFO might provide inadequate storage. We need to understand these parameters better.

The input and output interfaces of each FIFO register are pipeline interfaces. The input interface to each FIFO register has an integer destination and a token source. Whenever a FIFO register receives an integer at its input interface, it will produce a token on its input interface token source. These tokens may be discarded, used as completion signals for operations that put values into the FIFO registers, or couple the register file input interface to the output interface of some other unit.

The output interface for each register will have an integer source and a token destination. The register will release an integer value to its output interface source only upon receipt of a token at its output interface destination. Such tokens may come from a fixed token source, or from some pipeline input interface to which the register output couples.

DISCUSSION

Because the registers of the register file have pipeline interfaces, they can be coupled to other pipeline interfaces in the usual way. A pair of standing or counting moves placed between two such interfaces serves to communicate between them automatically. Thus, for example, a program can easily move multiple values from a memory control SHIP to or from a FIFO register with two standing or counting moves.

This document is a product of a collaboration between Sun Microsystems and the University of California at Berkeley. The ideas contained herein are freely available for any academic purpose.

Notice that the FIFO registers can also couple to each other. If eight entries in the FIFO prove inadequate, a pair of standing or counting moves between the output interface of one FIFO register and the input interface of another FIFO register will cause them to behave as a single FIFO register of twice the length. Such a compound FIFO register will, however, have a latency greater than twice the latency of a single FIFO register because there will be some switch fabric delay in the coupling.

SAVE AND RESTORE

The most troubling part of the register file design involves saving and restoring the content of the register file. To understand this problem, consider the content of the eight integer parts of a single FIFO register. Each such integer register may be empty, it may contain any valid integer, or it may contain some OOB value. How are we to store these possible contents in main memory and later restore them?

There is an OOB integer value representing termination of a string of integers. The single FIFO register might contain one or more such values. For example, a FIFO register might contain three integer strings, each of a single integer, separated by OOB values representing the “end” of each string. That content would leave two of the integer registers in the FIFO empty. Thus, we cannot use the OOB value representing the end of a string also to represent the empty state.

Thus, I see a need for multiple OOB values. We need at least: one as a string separator and one to represent an “empty” value. If we are able to store these OOB values in memory, we will be able to save and restore the entire FIFO register file.

The FIFO register file should discard any “empty” OOB values it receives at its input interface. However, how does the FIFO register file generate “empty” OOB values at its output interface?

A FILE DUMP INTERFACE

Suppose that for every four FIFO registers of the register file we provide an additional input output interface called the “dump” interface. I pick the four FIFO registers as the group size to make four separate groups. The dump interface will have one integer source and two token destinations; one destination will serve as part of a pipeline interface with the source. The other token destination will serve to initiate a dump of those four FIFO registers.

When a token arrives at the dump control destination, input to those four registers will be blocked and a dump started. There are $4 * 8 = 32$ integers involved in such a dump, some of which may be OOB values, and some of which might be empty. These 32 values will appear successively at the integer dump source using the usual pipeline communication convention. They will represent the content of the four FIFO registers taken in sequence. Thirty-two tokens must arrive at the dump interface token destination, one for each output to the dump integer output.

Thus to dump four FIFO registers to memory, one connects the dump interface to the memory write interface with a pipeline connection. Wait, such a pipeline connection is incapable of passing any string termination values that may appear in the FIFO register file. Thus, the pipeline connection will require 32 single move instructions in each direction.

A FILE RESTORE INTERFACE

File restore can use a similar input interface. The restore interface requires one integer destination and one token source to provide the usual pipeline interface. An additional token destination will serve to initiate a restore.

When a token arrives at the token destination, control will clear the four FIFO registers of that group and suspend any normal input or output from them. Then it will accept integer values and produce tokens in the usual way at the restore pipeline interface. The data received during this process will enter the four FIFO registers of that group in sequence, eight integers to each FIFO register. Notice that the “empty” OOB values will count towards the count of eight required before switching to the next FIFO register, but will be discarded rather than stored.

CRITICISM

In order to do save and restore, I introduced a new OOB integer type, namely “empty.” I find this troubling because I don’t understand the implications of storing the “empty” OOB value in a memory cell. For example, if we attempt to use the FIFO registers to copy a saved image of the FIFO registers, can we pass the image through the ordinary interface to FIFO registers, or must we use the dump and restore ports? Under what circumstances is an “empty” OOB value eliminated?

I can think of several alternatives. One alternative would produce a count of the number of occupied registers in the register file. The dump interface would produce a format with this count, possibly zero, leading the content of each FIFO register. The restore interface would understand this format. The count might be encoded as a unary number obtained by simply copying the occupancy signals from the FIFO into an integer value. The thirty-two registers of the FIFO group might produce a 32 bit occupied field.