serious obstacle to fully exploiting the advantages of asynchronous operation.

Two alternatives to register bypassing have been considered which deliver some of the benefits without impeding the asynchronous operation of the pipeline. These are described below as *register through-passing* and *last result re-use*.

4.1: Register through-passing

The design shown in figure 2 is very conservative in its timing for a write operation which clears a lock and thereby allows a read to proceed. A mechanism under consideration would, with the addition of a latch to the write enable logic, allow the lock to be cleared much earlier in the write process. This would use the register itself as the direct transmission medium from the write to the read bus, and would reduce the worst case delay for write-then-read from 40ns to 20ns.

4.2: Last result re-use

Another mechanism under consideration detects data dependencies at the decode stage. Each instruction leaves behind in the instruction decoder a record of the register its result will be sent to, and when the next instruction enters the decoder its operand addresses are compared with this record. When a match is found, the read operation is bypassed and the result is collected for operand use directly. The mechanism has no effect on the design of the register bank in figure 2 as it is manifested in additional logic elsewhere in the decode and execution paths.

This second mechanism has the best performance when it can be applied, but it has several limitations. On the ARM in particular, all instructions are executed conditionally, and an instruction which fails to pass the condition test will not produce a result. But by this time its successor may depend on that result. Therefore this mechanism must include logic to determine whether or not an instruction may be annulled, which adds considerably to the complexity of the design.

There is considerable scope for further innovation in the design of bypass mechanisms for asynchronous processors, and the ideas presented here are at a preliminary stage of conceptual development.

5: Conclusions

This paper describes an elegant and cost-effective solution to the requirement for read locking a register bank in an asynchronous pipelined processor. The pipelining of a processor's functionality introduces the problem of verifying that an instruction entering the pipeline does not have data dependencies on instructions currently under execution. The asynchronous nature of the pipeline makes the position of a particular instruction within it indeterminate so it is convenient to assemble register destinations in a FIFO to serve both as a write-back queue and as the basis of the locking mechanism.

In operation a read request not encountering a lock has immediate access to the register bank, and the instruction may proceed along the pipeline. When a read operation encounters a lock it is suspended indefinitely pending the return of the write-back value; this merely causes this read process to be delayed and does not impose delays on other asynchronous components of the processor.

The resulting design allows efficient VLSI implementation and handles asynchronous concurrent read and write operations (without recourse to arbiters) whilst maintaining coherent register behaviour.

6: Acknowledgments

The work described here has been carried out as part of ESPRIT project 5386 (the Open Microprocessor systems Initiative - Microprocessor Architecture Project, OMI-MAP), and the authors gratefully acknowledge this support. They are also grateful for additional specific support received during the course of this work from Acorn Computers Limited, Advanced RISC Machines Limited, VLSI Technology Limited and Compass Design Automation.

7: References

[1] Martin, A.J., Burns, S., Lee, T.K., Borkovie, D., Hazewindus, P.J.,
"The Design of an Asynchronous Microprocessor",
Advanced Research in VLSI : Proceedings of the Decennial Caltech Conference on VLSI, (1989) MIT Press, pp 351-373.

[2] Sutherland, I.E.,
"Micropipelines", Communications of the ACM,
Vol. 32, Number 6, June 1989, pp 720-738.

[3] Udding, J.T.,
"Classification and Composition of Delay-Insensitive Circuits", PhD Thesis, Dept. of Math. and C.S.,
Eindhoven Univ. of Technology, 1984.

[4] Furber, S.B.,
"VLSI RISC Architecture and Organization",
Marcel Dekker Inc., NewYork, 1989.

[5] Paver, N.C.,
"Condition Detection in Asynchronous Pipelines",
UK Patent Application No. 9114513, October 1991.

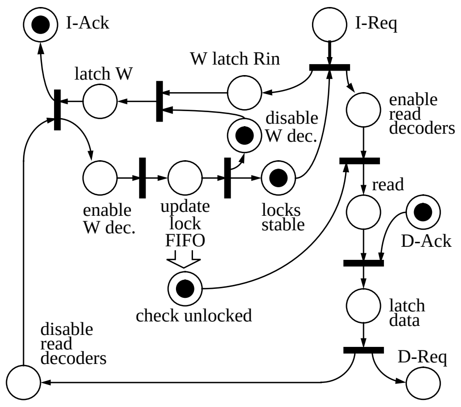

Figure 3: A Petri Net model of the read-lock sequencing

ter is unlocked for reading by removing its address from the FIFO. The write operation is self-timed by detecting the transitions on the word line with a wide dynamic OR gate, and the same circuit is used to ensure that writes are fully disabled before the write data is allowed to change.

3.3: Implementation details

The full design includes many features which have been omitted from figure 2 in order to clarify the design principles:

- Particular instructions may not require all three register addresses to be used, and logic is included to bypass any subset of them.
- The lock FIFO requires that write data arrives strictly in instruction order. Whilst the pipelined nature of the execution path ensures that this will apply to internal data sources, memory accesses are expected to arrive more slowly. Therefore two lock FIFOs are included in the design to allow internal data values to overtake external ones.
- Register bank accesses interact with accesses to special registers; in particular the program status register access control logic is integrated into the main register control structures.

Storing the decoded write address in the lock FIFO at first sight appears inefficient in terms of silicon area. However this allows the full stack of word control logic (the A, B and W decoders, the lock FIFO, the read lock gating and the write enable and completion logic) to be pitch-matched to the register cell block, and the efficiency of the resulting regular layout is very high.

The correct operation of the full circuit depends on appropriate implementation of the bounded delay constraints. An entry to the lock FIFO must propagate through the OR gate series and arrive at the read-lock gate earlier than any subsequent decoded read address. A visual inspection of the organization in figure 2 would suggest that this constraint is easily met, but a final confirmation based on delays using capacitive loads extracted from the physical layout is necessary.

3.4: Simulated performance

The design as described above has been laid out in CMOS VLSI and simulated at gate and transistor level. The performance of the design is summarised in table 1 for worst case conditions on a 1µm process; typical performance will be twice as fast.

path	delay
I-Req to D-Req	20 nS
I cycle time	40 nS
W cycle time	30 nS
W-Req to D-Req (register locked)	40 nS

Table1: Simulated delays

Simulation has demonstrated the correct operation of the design under varying conditions, but further work is required to extend this to a formal proof of correctness.

4: Register bypassing

Typical instruction streams display frequent use of the result of one instruction as an operand of the next. Such data dependencies between consecutive instructions can cause a significant reduction in throughput for typical code, compared with best case code without dependencies, if the result is only available to the next instruction after it has been written back to the register bank.

Clocked processors generally use register bypassing to allow a result to be re-used without incurring the register write-then-read penalty. The global clock ensures that different parts of the processor are operating at fixed relative times, so the result and operand addresses at two stages can be compared to activate the bypass when appropriate.

In an asynchronous processor there is no such fixed relationship between the timing of operations in different parts of the processor, so explicit synchronisation is necessary if a similar result and operand address comparison is to form the basis of a bypass mechanism. This synchronisation will have a cost in reduced throughput and, since it forces lock-step operation of at least two parts of the processor, it is a

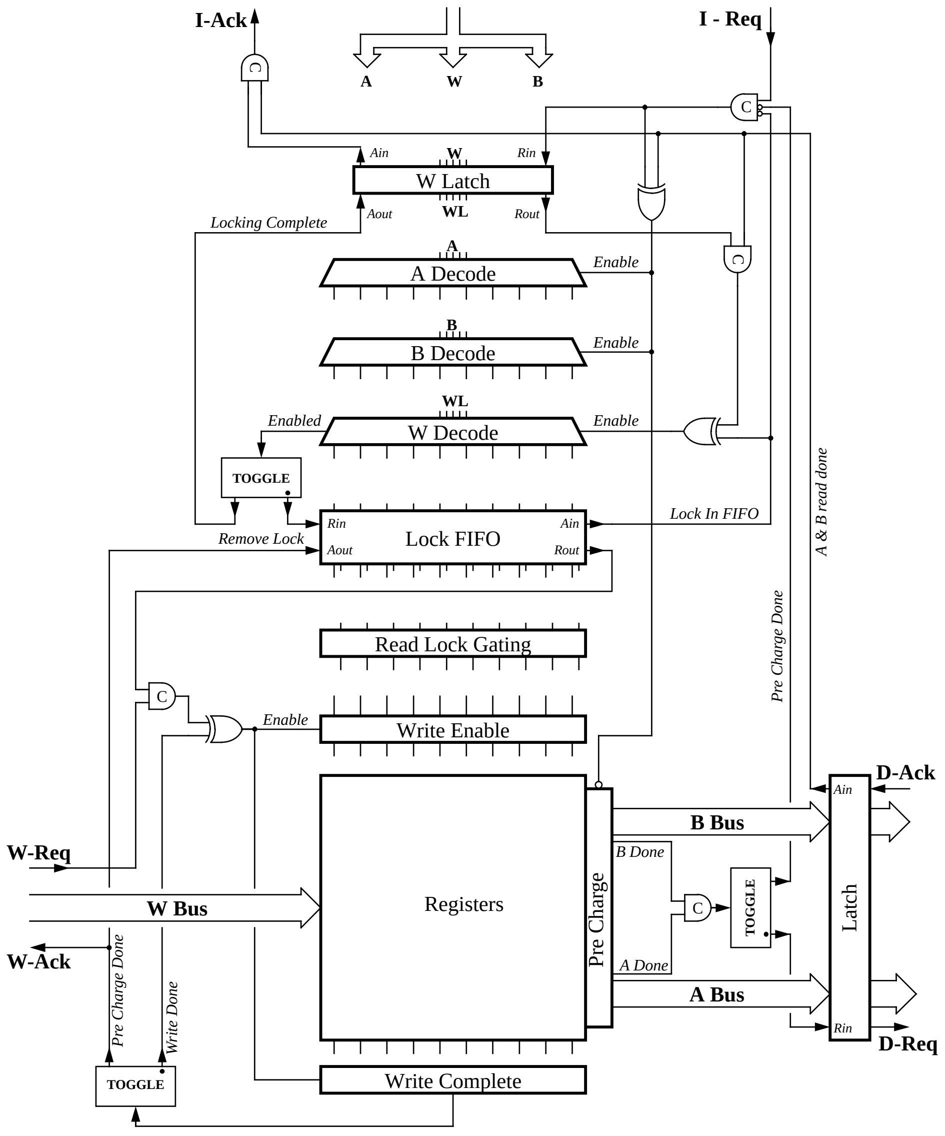

Figure 2: Organization of the register bank

cies in the read operation. Note particularly that the W decoder is disabled until the read has completed in order to ensure that no spurious lock indications are passed via the empty (and therefore transparent) stages in the lock FIFO; similarly it is disabled before the W latch is allowed to accept a new value. The next read is allowed to proceed as soon as the locks are stable, since any transient caused by the slow disabling of the W decoder will cause at worst a delay in the read operation, never an incorrect action.

The critical path in the register bank (from I-Req to D-Req) has the minimum number of dependencies on internal operations; this defines the register access latency of the design. The cycle time will include this and the slowest of three independent recovery routes:

- The supply of the next instruction.
- The completion of the locking operation.
- The read bus precharge time
  (omitted from figure 3 for clarity).

In general it is expected that the first of the above will be the critical path in determining the register bank cycle time.

3.2: Write operations

A write data value (signalled on W-Req in figure 2) is paired with the decoded write address at the output of the lock FIFO. The appropriate write word line is then enabled and the data written, following which the destination regisfunctional units are employed. The destinations of results to be returned to the register bank are therefore simply queued and used on a First-In First-Out basis to write results to the required locations. This FIFO holds the identifiers of all locations which are subject to alteration by instructions currently in the processing pipeline and can therefore form the basis of a locking mechanism to deny premature access.

To establish whether a read access to a register is permitted requires interrogation of the “Lock FIFO” contents. This FIFO is implemented with identifiers which are represented as decoded values; each entry has a single bit set in a field whose size matches the number of registers in the bank. Using such a representation reduces the lock interrogation process to that of determining whether a bit is set in the column of the FIFO corresponding to the register to be read [5].

The Lock FIFO fulfils two requirements: firstly, it provides an indication of whether a register is locked and, secondly, its last stage contains the decoded address for the next write operation. A simplified version of the Lock FIFO is shown in figure 1 for a bank of three registers with a 2-stage pipeline, together with the associated read and write logic.

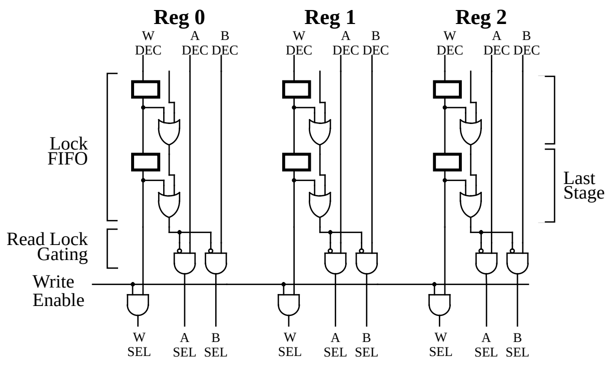

Figure 1: Lock FIFO, read gate and write enable logic

Certain constraints must be imposed to ensure that this design will function correctly. The operation of the FIFO must be organised so that a bit appears in a stage before it is removed from its predecessor to avoid any transient removal of the ‘locked’ condition. An empty stage must also present an unlocked state so that it takes no part in the locking process. Furthermore, new values must not be allowed to enter the FIFO while the lock status is being inspected to avoid glitches on the lock outputs.

The signalling protocols of the Sutherland micropipeline ensure that data is not removed from a pipeline stage until it has been established in the succeeding stage. Empty pipeline stages have a transparent property so that, provided the input of the pipeline is held inactive when no entry is being made, empty stages will naturally assume an unlocked state. Ensuring that there is no conflict between inspecting the lock status and the entering of a new destination is left to a higher level of control.

3: Asynchronous register bank design

The overall organization of the register bank is shown in figure 2. The interfaces use a bundled-data convention with transition signalling. Internally the design employs a combination of two-phase and four-phase techniques (see e.g. [2]), the latter being well matched to the precharge-active cycle of the dynamic circuits used in the basic register cell.

3.1: Read and lock operations

A new instruction has its availability signalled by I-Req, and presents two register addresses to be read (A and B) and a register address to be written once the datapath result is available (W). I-Req is stalled until the register bank is ready to start a new read operation, and then the read decoders are enabled. Concurrently with the read address decoding, the write register address is latched (in W Latch).

The decoded read addresses now present enables for the selected registers which are gated with the locked register information. A read of an unlocked register will proceed, whereas a read of a locked register will stall at this point until a write operation clears the register lock.

The register read circuitry uses dynamic techniques to minimise the cell size, with charge retention circuits to give pseudo-static operation. A thirty-third bit line gives a matched completion signal, and when both values are available they are latched and passed to the execution path (via the D-Req signal) which can begin to process the data with no further delay.

Once the data has been latched, the read decoders are disabled and the read bus precharge is turned on to prepare for the next access. Normally the write address will be latched well before this time, and the instruction acknowledge (I-Ack) is issued so that a new instruction can be prepared during the register recovery time.

The write decoder is disabled during the read operation to present inactive inputs to the lock FIFO. Once the read data is latched, the write decoder is enabled and the destination register is locked. As soon as the lock FIFO has accepted the new address, a new instruction may be allowed to start its read operation. The lock logic will continue by disabling the write decoder and it will then free the write address latch for the next value.

The read-lock sequencing is illustrated in Petri Net form in figure 3, which shows the critical sequential dependen-

Register Locking in an Asynchronous Microprocessor

N.C. Paver, P. Day, S.B. Furber, J.D. Garside, J.V. Woods

Department of Computer Science, The University,
Oxford Road, Manchester, M13 9PL, U.K.

Abstract

A high performance register bank is a central component of a RISC processor. A novel register bank design has been developed, as an integral part of a self-timed implementation of a commercial RISC microprocessor, to address the problem of register interlocking in an asynchronous micropipelined execution unit.

The challenge in an asynchronous design is to maintain coherent register operation while allowing concurrent read and write accesses with arbitrary timing. The solution presented here includes a novel arbiter-free locking mechanism which enables efficient read operations in the presence of multiple pending write operations.

1: Introduction

The growth in demand for high performance portable computing equipment has led to a resurgence of interest in asynchronous logic design techniques. In order to investigate the power saving potential of asynchronous approaches to CMOS design, a self-timed implementation of the ARM microprocessor is being developed as a commercially realistic technology demonstrator. Earlier work [1] has shown the feasibility of building a complete asynchronous microprocessor; the current project addresses the detailed problems associated with implementing a commercial architecture with the specific goal of minimising power consumption.

The methodology being applied is based on Sutherland's "Micropipelines" [2], a bundled-data, bounded-delay model. Here, local timing signals are transmitted with a 'bundle' of data bits whose timing is constrained to ensure correct operation. This technique - rather than a purely delay-insensitive model [3] - was chosen for its economy in silicon area and its potential for low electrical power consumption. The micropipeline approach is somewhat less 'pure' than other approaches to the construction of asynchronous systems because delays in the circuit must be managed quantitatively; however these delays can be modelled and characterised in a similar manner to the critical path analysis used in the design of synchronous circuits.

The design of the processor can be decomposed into a few major structural elements, one being the register bank. The ARM register bank contains thirty one registers, of which sixteen are available to the programmer at a given time. All but one of these registers are general purpose and orthogonal; the implementation of the ARM register bank described here is therefore applicable to asynchronous implementations of other RISC processors.

2: Register locking

The ARM architecture [4] defines a register-based RISC processor in which arithmetic operations require two operands to be read from the register bank and a single result value to be returned. In existing synchronous implementations of the architecture instruction *execution* is not pipelined (execution is a single stage of the Fetch - Decode - Execute pipeline) and an arithmetic operation is completed within a single clock cycle. In the asynchronous implementation instruction execution is decomposed into a number of pipeline stages. This concurrent execution improves performance but introduces the problem of data dependency.

Correct operation in a pipelined processor requires that data dependencies between instructions are respected; this may be achieved by ensuring that a location subject to modification cannot be accessed until the pending write operation has completed. This process is termed 'locking'.

2.1: The lock FIFO

Pipelined operational parallelism does not change the ordering of generated results as may happen when parallel