Chapter Seven
Conclusion

This thesis has demonstrated a fault-tolerance method for an asynchronous processor. The register reservation mechanism guarantees mutual exclusion on the registers and enables concurrency for independent instructions. The instruction log and delayed write buffers provide temporary storage for un-validated data so that permanent state elements are not affected until verification is finished. Because execution of instructions may complete out of order in an asynchronous environment, sequence numbers are used to track dependencies at the instruction level, and instructions which need to be rolled back can be properly identified and undone.

The performance of a synchronous system is penalized because the clock rate is limited by the longest path of all pipeline stages, and time may be wasted in many stages. An asynchronous system requires only as much time as necessary to complete a task, and therefore, it achieves an *average* processing speed, not the worst case. Asynchronous design also allows the system to be very modular because there is no global constraint like the clock. Blocks of different speeds may be mixed and matched without altering the functionality of a system, as long as the handshake protocol is not violated. However, some resources may be well under-utilized if the system is not balanced properly, and the overhead for handshaking must be considered.

One of the difficulties with a fully asynchronous design is generating the handshake signals in the correct sequence. A state machine in the traditional synchronous sense is not available because there is no clock. Certain sequencing can be forced by introducing delay elements, as done in some circuits in Chapter 6, but their accuracy and reliability in different conditions (electrical, thermal, and IC process

variations) are the drawbacks. Too short of a delay makes the circuit non-functional, but the performance suffers if the delay is longer than necessary. CAD tools should really be used to assist the design of complex asynchronous systems, in terms of handshake circuit synthesis and analysis.

As discussed in Chapter 2, delay-insensitive circuits guarantee correct asynchronous operations because no assumption is made on wire delays. However, the increased circuit complexity and chip area can be significant, and the larger physical size does mean increased propagation delays. Since gate and wire delays can be controlled locally without much difficulty, self-timed but delay-dependent circuits can be applied in small blocks. Proper trade-offs between the two design methods should lead to a better implementation than either one alone.

One of the goals at the beginning of this research was to design an equivalent of micro rollback in an asynchronous environment. Micro rollback works at a finer granularity than instruction retry, and as a result, the number of repeated operations is reduced. For example, it is not necessary to fetch an instruction again if the error of the same instruction occurs in the execution stage. In an asynchronous processor, however, each unit cannot simply go back a few steps because each one operates at its own speed. Dependency tracking probably has to be based on individual operations rather than instruction sequence numbers, and if more than one dependency has to be kept [Stro85], then it may become very difficult to manage. Furthermore, each functional unit (such as the ALU) may need to keep a history log so that previous data can be played back for rollback.

Some areas of this project can be improved and expanded. In AMPIRE, the instruction and data memories are completely separate units so that they are easier to be controlled. In a real system, though, a single memory device usually holds both

instructions and data. An arbiter has to be added to coordinate instruction and data access, and the instruction fetch process has to be changed because now the IMEM portion can be modified. Also, many faults cannot be detected and corrected with a single processor. If two asynchronous processors are to be connected as a master/slave pair like the UCLA Mirror Processor, then additional handshake mechanism has to be added for synchronization, which is another layer of overhead. Since AMPIRE has not been physically implemented, the impact of the addition of fault-tolerant features cannot be quantitatively measured. However, based on the architecture of parallel data verification, results should be similar to those of the Mirror Processor: high overhead in chip area but insignificant reduction in performance.