Chapter Two
Previous Work

2.1. Handshake Protocols

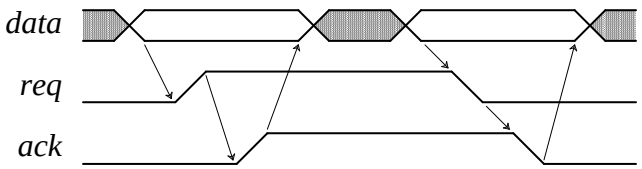

Figure 2.1: 2-Phase Handshake Protocol

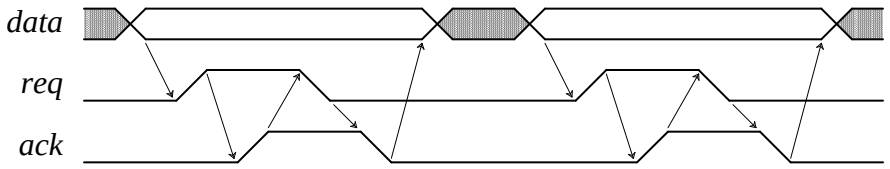

Figure 2.2: 4-Phase Handshake Protocol

Whereas a synchronous system initiates a task cycle with clocks, an asynchronous system uses handshake signals to start and stop an activity. The 2-phase and 4-phase self-timed signaling conventions in [Seit80] are presented in Figures 2.1 and 2.2 in a slightly modified form. For both cases, the request is sent after data becomes stable. After the receiver finishes processing the data, an acknowledgement is sent back, and then data is released. 4-phase handshake is level-sensitive, and it may be slower because an extra trip is required to disable the *req* and *ack* signals. However, 2-phase handshake needs edge detectors, which results in more complicated circuitry. Therefore, 4-phase signaling is generally used for local communication, leaving 2-phase signaling for long distance transactions. Power wise, 2-phase has the advantage of using half as many signal transitions, and the energy savings may be significant if the

interconnection wires are long.

Note that in Figures 2.1 and 2.2, data must be stable before the request signal can be activated. This is true at the transmitter, but if data and handshake signals are routed differently, this ordering may not be preserved at the receiver, and therefore violating the protocol. This type of handshake is valid only in *equipotential regions*, unless routing and delays can be carefully controlled. An equipotential region is an area small enough such that delay through the wire is small compared to signal rise and fall times [Seit80]. For communication outside of an equipotential region, a *delay-insensitive* protocol should be used to maintain correct self-timed operations.

Delay-insensitive design is a subset of the class of self-timed circuits. One way to achieve delay-insensitivity is by using double-rail encoding [Seit80], in which the (data,$\overline{\text{data}}$) pair is: 00 for undefined, 10 for one, and 01 for zero. 11 is not allowed, and only one of the two signals may switch at any time. A handshake protocol using this encoding is shown in Figure 2.3. The first cycle transfers a one, and the second cycle transfers a zero. If a request signal is needed at the receiver end, it can be easily generated by ORing the double-rail data lines. Figure 2.4 is a circuit that uses a *C–element* to merge the multiple internal request signals. Muller C-elements are often used in self-timed systems and have the following characteristics: when all inputs are 1, the output becomes 1, and when all inputs are 0, the output becomes 0. Otherwise, the output remains in its previous state. C-element implementations will be discussed in section 6.1.

Delay-insensitive circuit removes a timing constraint, but it also carries a high price tag. For single-rail plus request and acknowledgement protocols, a data bus of any width only has to add two wires for handshaking. The double-rail method, while more reliable, requires twice as many signal lines. Therefore, the trade-off is based on

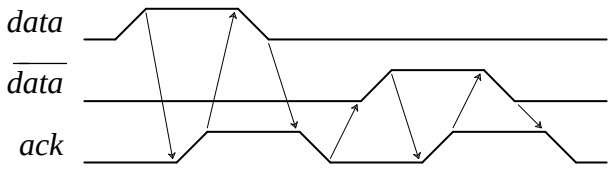

Figure 2.3: Delay-Insensitive 4-Phase Handshake Protocol

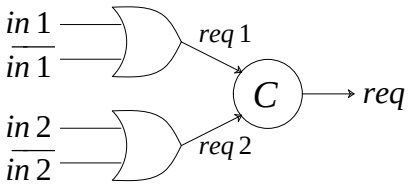

Figure 2.4: Request Generation for Double-Rail Data

how much control the designer has on signal delays and wire/block placement. Conversion between single-rail and double-rail signaling is quite straight forward, and a circuit can be found on [Seit80, p. 257].

2.2. Asynchronous Processors and Circuits

The asynchronous processor developed at Caltech is entirely delay-insensitive, with the exception of *isochronic forks* [Mart89]. An isochronic fork is the distribution of a signal to several receivers, and the differences in delays are assumed to be small compared to gate delays, like an equipotential region. It is a general purpose 16-bit microprocessor with load-store architecture and separate instruction and data memories, with double-rail encoding, 4-phase handshaking as the communication protocol. The performance profile for the 2 μm version is shown in Figure 2.5. At room temperature, the chip is functional with the supply voltage as low as 0.35V, and the speed reaches 30 MIPS at 12V when the chip is submerged in liquid nitrogen. All these performance variations occur without any clock adjustments, since there is no clock.

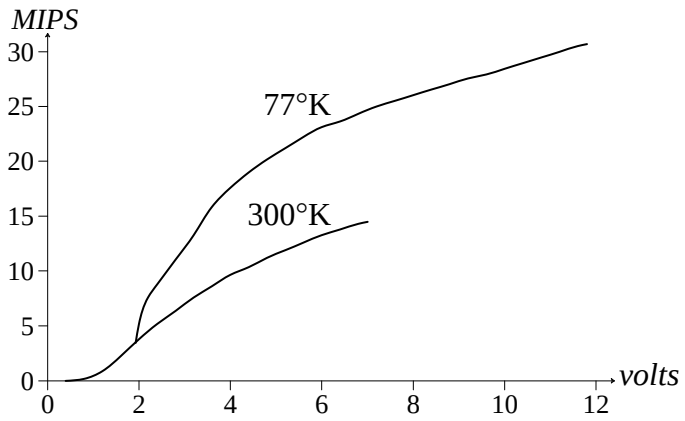

Figure 2.5: MIPS as a Function of Supply Voltage for Caltech Processor

Another fully asynchronous chip is a digital signal processor (DSP) designed at UC Berkeley [Jaco90]. A ripple-carry adder and an iterative multiplier are used to save chip area and to take advantage of the self-timed circuits. Since an instruction cannot utilize both the shifter and the multiplier at the same time, these two units are placed in the same pipeline stage. Therefore, an instruction cycle time is highly dependent on the instruction and the data being executed. Table 2.1 lists the average instruction cycle time at various supply voltages.

Vdd	Shift	Multiply
3.6 V	105 ns	440 ns
5.0 V	73 ns	337 ns
7.0 V	55 ns	260 ns

Table 2.1: Average Instruction Periods for Berkeley DSP

The DSP chip is self-timed, but not delay-insensitive like the Caltech processor. Single-rail data and request signals are used, as shown in the processor handshake model in Figure 2.6. After the request is received, the register is clocked to latch the data. Since there is no feedback from the register about its completion, an assumption is made on the delay before the signal I (initialize) is raised to start an evaluation cycle.

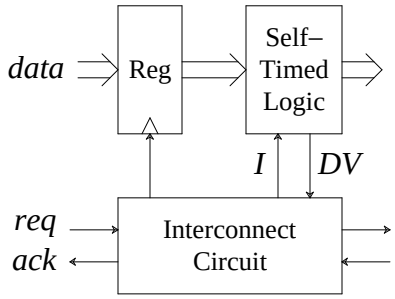

Figure 2.6: Handshake Model for Berkeley DSP

After the self-timed logic block is finished, the data valid *DV* signal is sent back to notify the interconnect circuit. A problem was encountered in the initial layout that resulted in a very long wire for the register clock. The additional delay was long enough to cause the logic block to start evaluation before the data bits were settled, but a change in the floorplan solved the problem. This case demonstrates that by using the more efficient, delay-dependent circuits, some freedom of block placement is sacrificed.

As mentioned in Chapter 1, handshake circuits are difficult to design manually because all events have to occur in the correct sequence without, ironically, a clock. Both processors discussed in this section started with high-level signal descriptions, and then the handshake/control circuits were synthesized with CAD tools. Even with synchronous systems, the control blocks in most processors nowadays are built through hardware synthesis. The various methods are too complex to be covered in this thesis. Please see these references for details [Burn87, Mart85, Mart86, Meng89, Moln85].

An asynchronous system is not complete without self-timed memory. A self-timed static RAM is discussed in [Fran83]. The memory array uses conventional six-transistor static RAM cells, but circuitry is added to support the additional handshake requirements. Since the RAM cell and sense amplifier already have differential bit

lines, similar to double-rail encoding discussed earlier, generating the completion signal is a very natural extension. The external interface has additional request and acknowledge lines to be connected to other asynchronous devices, such as a processor. Only 5.2% of the total chip area is occupied by the self-timed completion detectors.

2.3. Micro Rollback

Micro rollback (in a synchronous system) works by taking snapshots of the state of a module, and when an error is detected, a valid state is restored using the saved information. Figure 2.7 is an illustration of micro rollback [Tami90a]. Error is detected a few cycles later because checkers operate in parallel with the functional units, in order to minimize performance degradation due to data verification. Micro rollback differs from *instruction retry* [Ciac81] because it is based on clock cycles rather than full instructions. As a result, micro rollback can be independently implemented in each module that uses the clock to advance its state, regardless of its function or the pipeline structure.

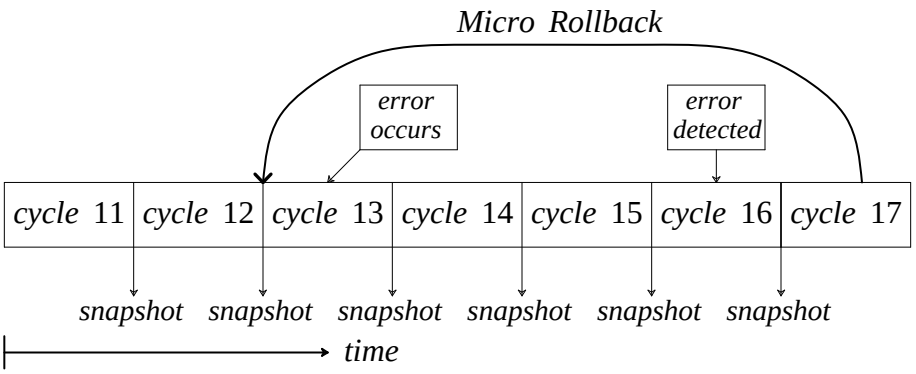

Figure 2.7: Micro Rollback, Restoring a Saved Snapshot

Storing the state of a simple register can be accomplished by adding a controller and connecting several register elements in a FIFO fashion. However, for a large

register file, it is not practical to duplicate the entire block several times to allow rolling back multiple cycles. Since only one register (or a few registers, depending on the instruction set) can be written per clock cycle, a delayed write buffer (DWB) is used to hold the data targeted for the register file until check is complete, as shown in Figure 2.8 [Tami90a].

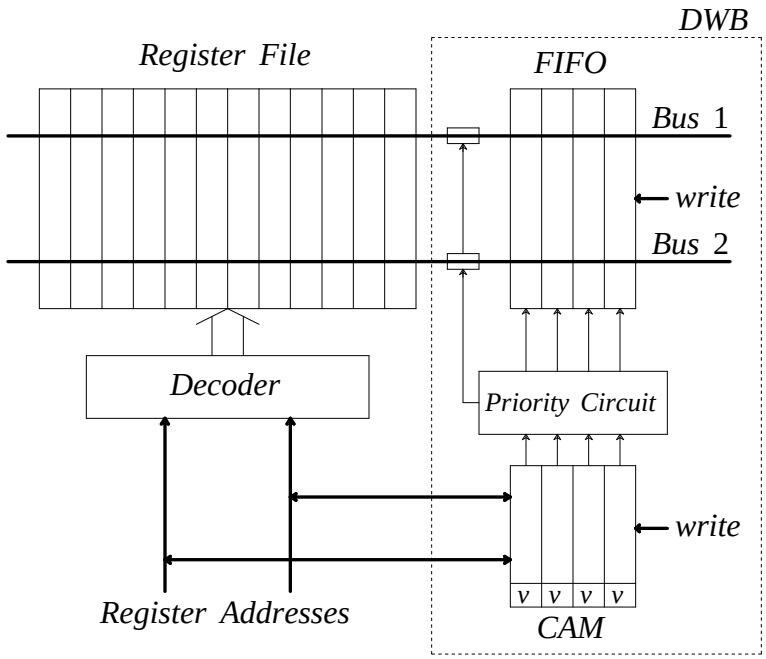

Figure 2.8: Register File with Support for Micro Rollback

The content-addressable memory (CAM) contains the destination register addresses, and a valid bit indicates that the corresponding FIFO buffer has a valid data to be written to the register file. If a clock cycle does not update the register file, the DWB is simply shifted to the left without setting the valid bit, and when a valid data is shifted out of the DWB, it is *committed* to the register file. Therefore, the depth of DWB determines the maximum number of cycles that can be rolled back, and verification must be done within that time. When an error is detected, the appropriate valid bits are cleared, and rollback is achieved because the state changes never reached

the register file. The priority circuit is necessary to retrieve the most recent register data, even if it has not been written to the register file yet, so that other instructions dependent on the data are not blocked from execution.

The UCLA Mirror Processor [Tami90b] is a fault-tolerant RISC microprocessor that is capable of micro rollback. In addition to the on-chip parity checkers and DWBs for error detection and recovery, two processors can operate in lock-step, one master and one slave, comparing both external signals and internal *signatures*. It is very expensive to route tens if not hundreds of internal signals to the pins. Therefore, interleaved parity bits of the desired signals, called signatures, are generated with chains of switching XOR cells [Trem89], and the condensed data is then used for comparison. When a mismatch is found, both processors are rolled back the same number of cycles. However, certain transient errors, such as a fault in the register file, cannot be corrected with DWBs alone. Under these conditions, the faulty data in one processor is replaced with the correct one from the other processor. If both processors have errors in the same location, then a higher level recovery scheme is necessary.

AMPIRE supports instruction retry, not micro rollback, but DWBs are still used to postpone write operations to the register file and the data memory, as will be seen later in this report. All components of the DWB are present: FIFO, CAM, and the priority circuit, except that they are replaced by their asynchronous counterparts.