Is Your 64K Dynamic RAM Refresh Scheme Killing Your μP Performance?

by Dick Brunner, Motorola, Inc., MOSIC Div., Austin, TX

One of the major applications for 64K dynamic random access memories (DRAMs) is main memory for μPs and computers. Their high density, low power and fast access make them ideal for this application. To accomplish this high density and low power, the 64K dynamic RAM employs a storage element consisting of a select transistor and storage capacitor. Digital information (logic zeros and ones) are stored on these storage capacitors by the presence or absence of charge. This type of storage mechanism requires a periodic update of the capacitor charge state to insure that the stored information is not lost due to leakage current.

To insure the capacitor's charge state, a periodic refresh cycle is performed on all memory cells; which for most 64K DRAMs is accomplished with 128 refresh cycles every 2ms.

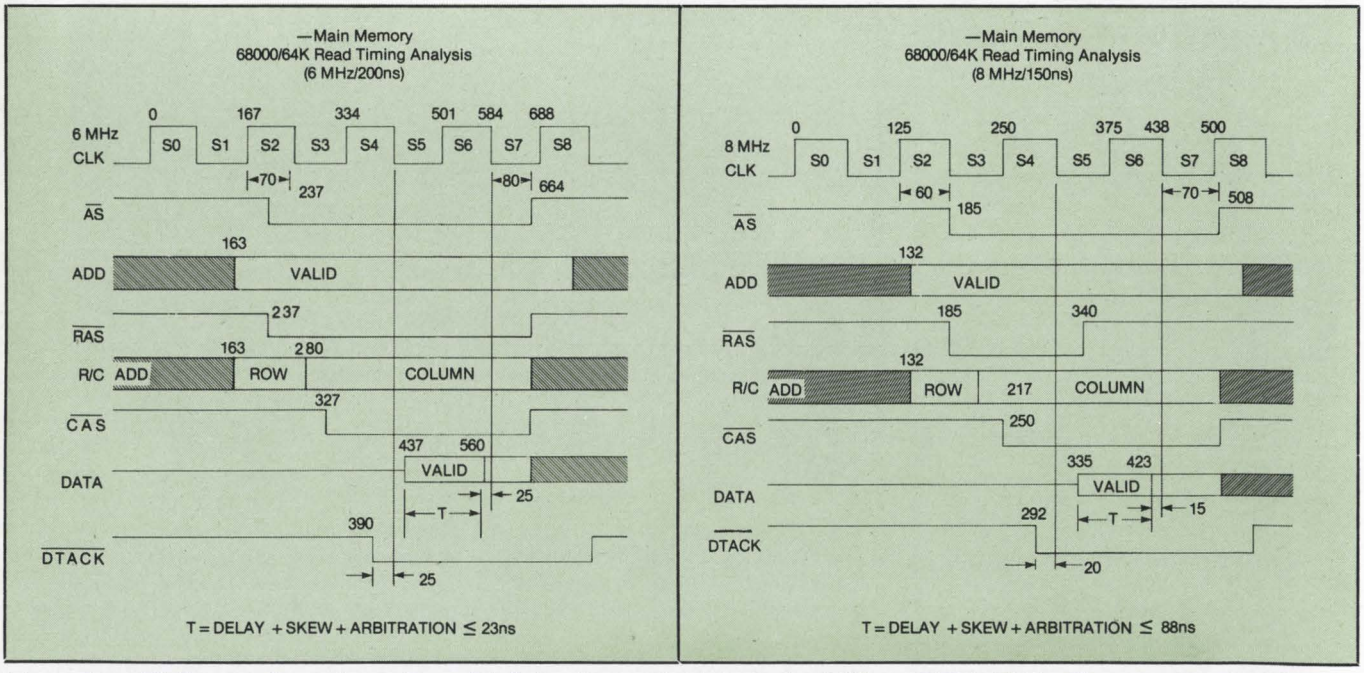
Potential increase in data throughput is 22% if MPU refresh control is used vs. that with memory controlled refresh.

On the surface, it appears that implementation of the refresh cycles along with the normal μP or computer cycles would be straightforward and have minimal effect on data throughput. However, a close look reveals that implementation of these refresh cycles dramatically impacts data throughput of all high performance asynchronous μPs and computers.

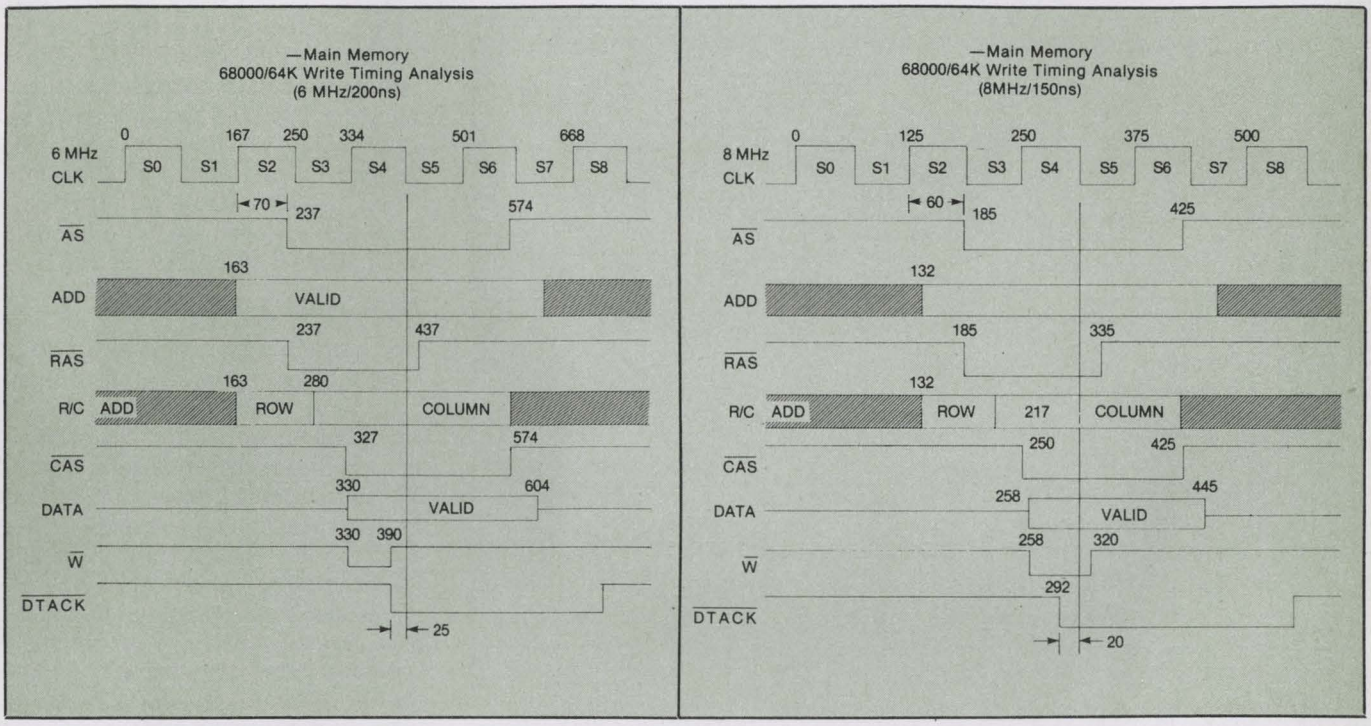
Slave Peripheral

Main memory is not generally regarded as a peripheral, but is in actuality. In fact, the memory is a slave peripheral that must interface with at least two master controllers—μP or computer and a refresh controller which will insure that the memory maintain valid data. Since both controllers are asynchronous to each other, there will be brief periods in time when both are trying to access the memory at the same time.

To insure an orderly synchronization of these signals during the periods of contention, arbitration logic is necessary. When two signals of dif-



Figures 1a and 1b illustrate the maximum data valid window for a read cycle for both a 6MHz and 8MHz 68000, with respect to both a 200ns and 150ns access part.



Figures 2a and 2b show the comparable worst case timings for a write cycle. Note that data valid does not occur at the beginning of the cycle.

ferent frequencies are to be synchronized, there will be periods when input parameters are violated and the logic device output will enter an undefined state for a period of time. Eventually the logic device output will settle into a stable state. For an S74 type latch this period can be as long as 75ns.

This extended period of time for a bi-stable latch to settle out can be sufficient for a high performance μP system to result in a design that requires the μP to generate wait states on all memory cycles. To illustrate this, let us examine the bus timing of the 68000 with respect to the timing requirements of the MCM6665 (64K DRAM). **Figures 1a and 1b** illustrate the maximum data valid window for a read cyle for both a 6MHz and 8MHz 68000, assuming no wait states, with respect to both a 200ns and 150ns access part. Note from these figures that for a worst case 68000 read cycle time, and assuming zero logic delays, that the minimum to maximum data valid window T is only 123ns and 88ns, respectively. If arbitration is required in the memory control logic to accomodate the refresh cycles, then the available time left for bus and timing logic is only 48ns and 13ns, respectively. Even using Schottky logic it would be virtually impossible to have a conservative design that would not require wait states during the read cycle.

The comparable worst case timings for a write cycle are given in **Figures 2a and 2b.** Note for a write cycle that the

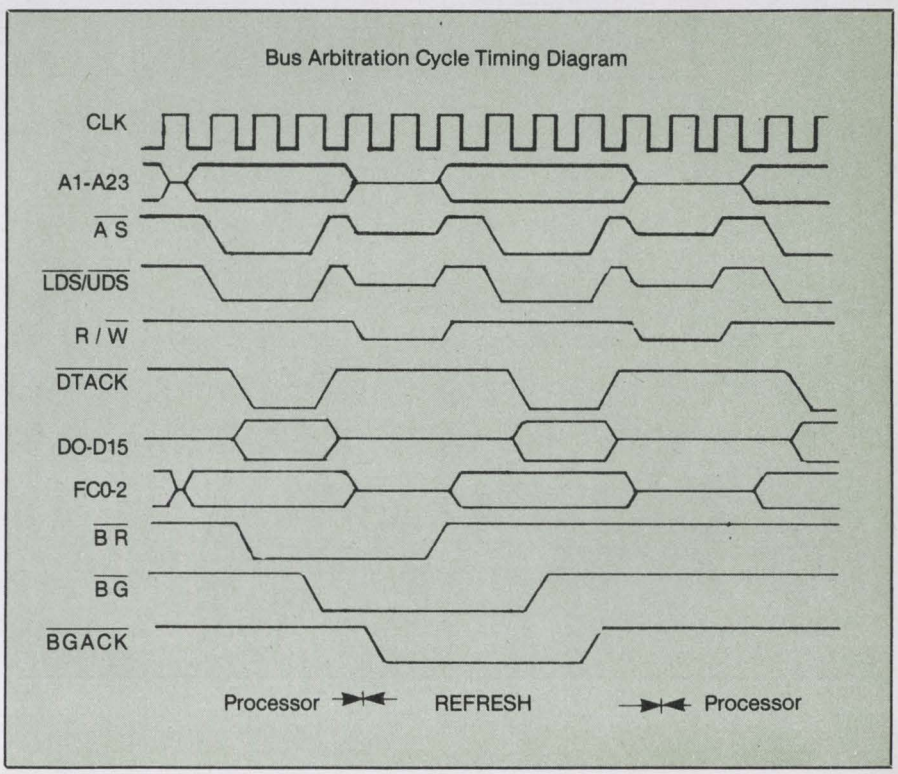


Figure 3: The 68000 can handle the bus arbitration of the refresh cycle requests.

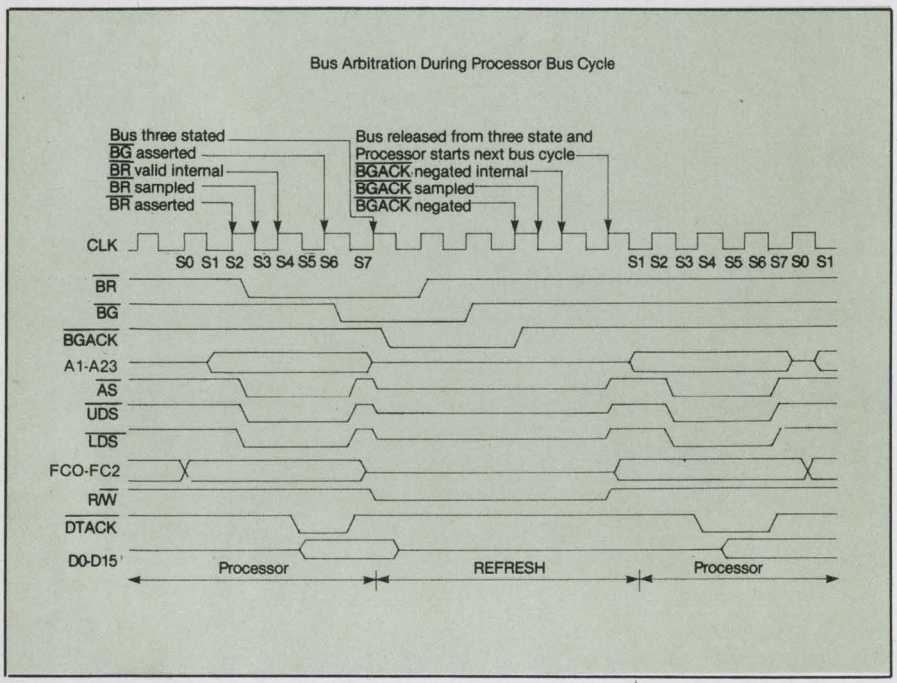


Figure 4: The 68000 acknowledges the refresh cycle and the refresh controller originals that it has control of the bus.

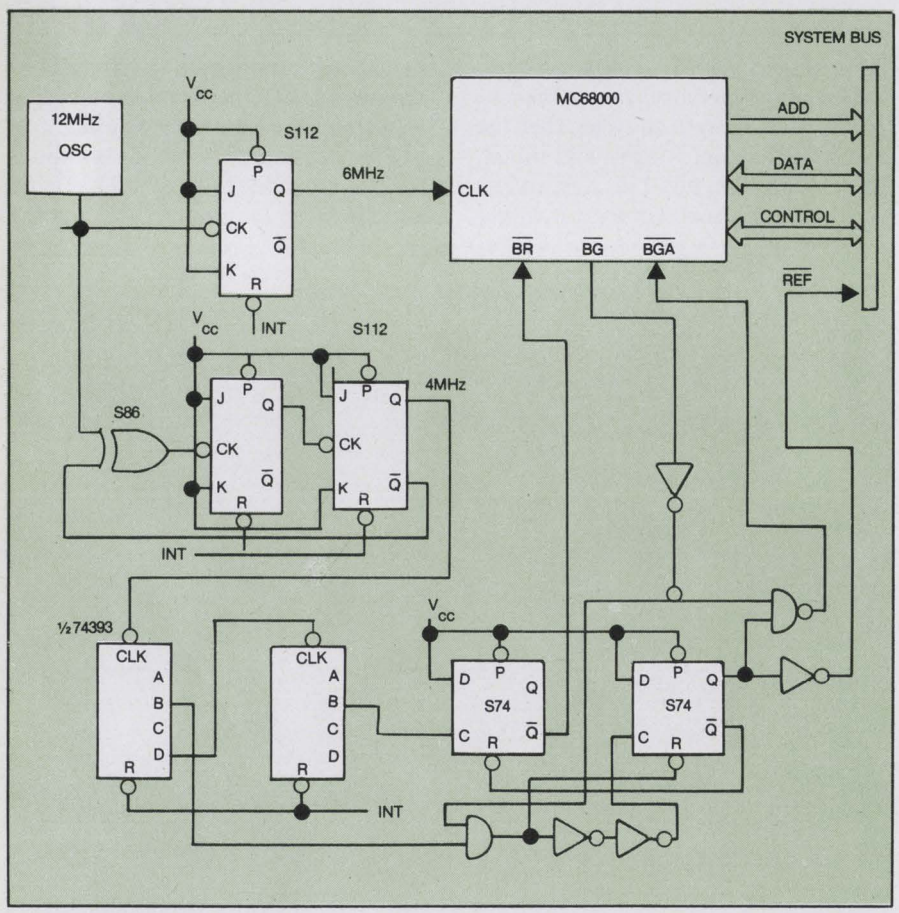


Figure 5: Illustrating a refresh control scheme for a 6MHz μP.

timing of the 68000 and the 64K are more compatible; hence, they will not require wait states to accomodate arbitration. It should be noted however, that the data valid does not occur at the beginning of the cycle ($\overline{AS}$ going True); hence, to minimize the write cycle time and accommodate the late data valid, a late write cycle is required.

Moving onto the μP Card

Fortunately, there is an alternative system design approach that will accommodate the dynamic refresh without having to suffer the arbitration overhead on every memory cycle. The refresh control logic can be moved to the μP card and integrated into the control and clock circuitry of the μP. Since the 68000 is an asynchronous μP, it was designed to easily accomodate asynchronous interrupts. It has internal logic to arbitrate incoming asynchronous interrupt requests such as refresh. And, noted in **Figure 3,** the 68000 can handle, very time efficiently, the bus arbitration of the refresh cycle requests. Through the bus arbitration logic of the 68000, a refresh cycle request can be initiated by driving the bus request ($\overline{BR}$) input low. The 68000 will acknowledge the refresh cycle when bus grant ($\overline{BG}$) is asserted. The refresh controller signals that it has control of the bus with a bus grant acknowledge ($\overline{BGACK}$) signal: this sequence is illustrated in **Figure 4.** The above refresh control scheme for a 6 MHz μP can be accomplished with the logic design given in **Figure 5;** the system design for the μP/memory interface and refresh logic is illustrated in **Figure 6.**

This μP/memory refresh scheme will require only two μP read cycles to accomplish the refresh cycle. The following analysis gives the maximum possible data throughput possible with MPU refresh vs that possible with a wait state for every memory cycle:

μP Refresh Control (memory cycle = 668ns). Refresh Period = 16 μs. Maximun number of cycles possible during this period is 24. Total μP cycles possible is 22.

Memory Refresh Control (memory cycle = 835ns; assume one wait state for each cycle). Refresh Period = 16μs. Maximum number of cycles possible during this period is 19. Total μP cycles possible is 18.

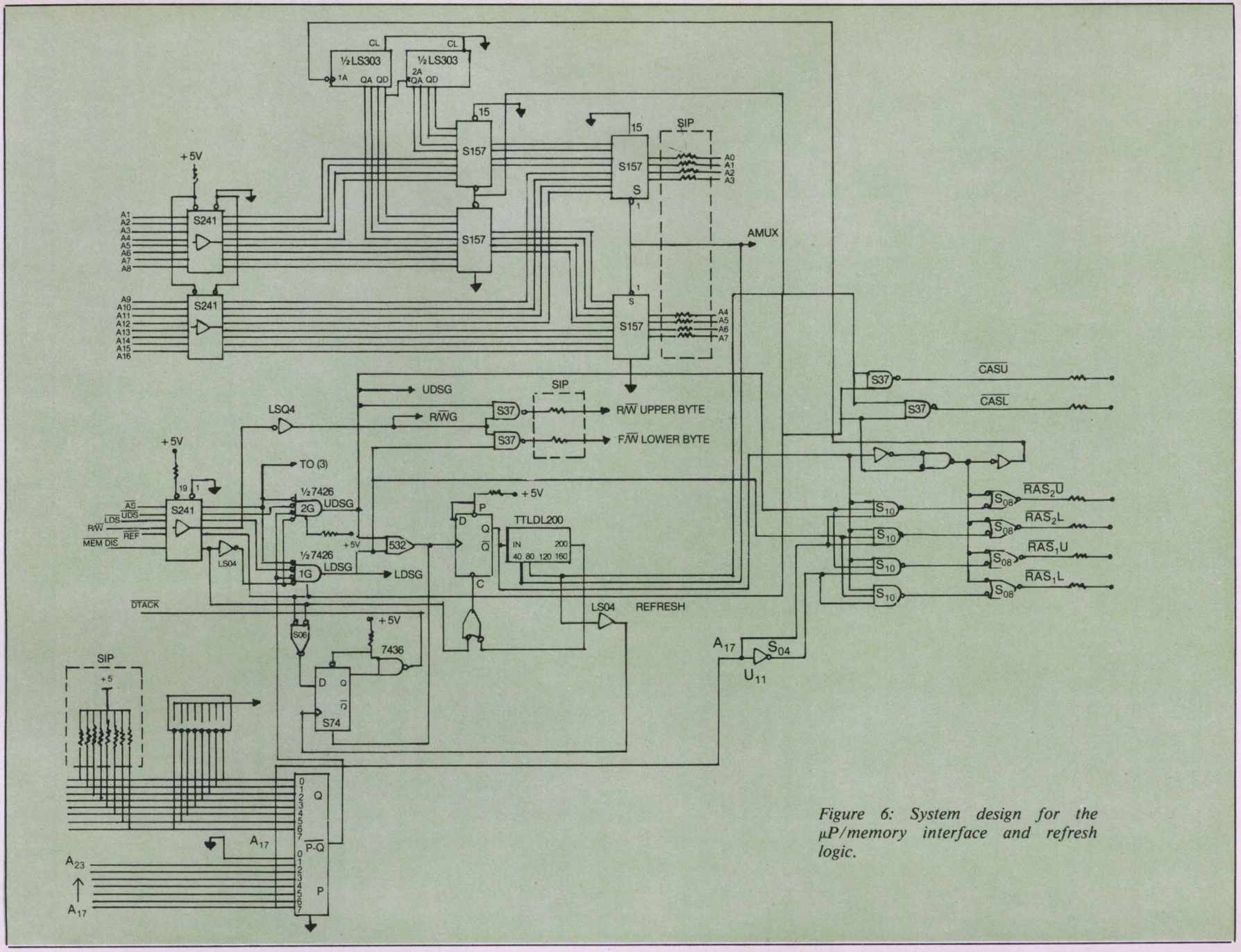


Figure 6: System design for the μP/memory interface and refresh logic.