**Synchronous SRAM Controller**

**Introduction**

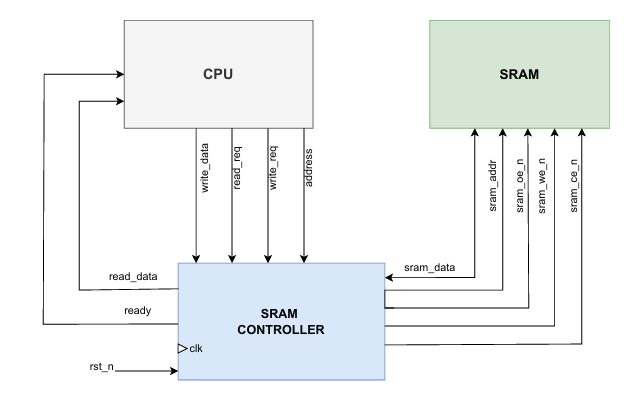
Static Random Access Memory (SRAM) is a fast, volatile memory technology. Unlike DRAM, it does not require periodic refresh; it retains data as long as power is applied. The device considered here is a 32K×16 synchronous SRAM, which provides 32,768 word locations addressed by A[14:0] and transfers data on a 16‑bit bus [15:0]. As a synchronous part, it samples address and control signals on a clock edge and produces read data after a defined clock‑to‑out delay.

A dedicated controller is required because the raw SRAM interface does not manage sequencing or bus direction. The controller ensures correct assertion of chip‑enable and read/write and safe control of the bidirectional data bus so that only one side drives it at any time

**Interface Overview**

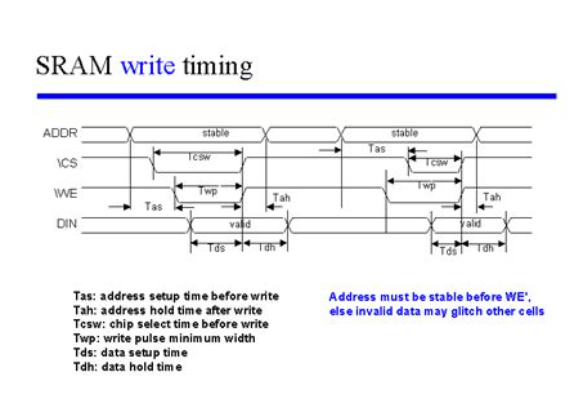
On the **system side**, my controller exposes a minimal command interface: read\_req, write\_req, a 15-bit address, a 16-bit write\_data, a 16-bit read\_data, and a ready flag. A request is accepted only when ready is high; the intended usage is to pulse exactly one of the request signals for one clock, then wait for ready to return high before issuing the next command. This makes each access atomic.

On the **memory side**, the controller drives sram\_addr[14:0] and the active-low controls sram\_ce\_n (chip enable), sram\_oe\_n (output enable), and sram\_we\_n (write enable), and connects to the bidirectional sram\_data[15:0]. During reads the SRAM must be the only driver of sram\_data; during writes the controller must be the only driver. Between operations the bus is intentionally left high-impedance to provide safe turnaround between directions.



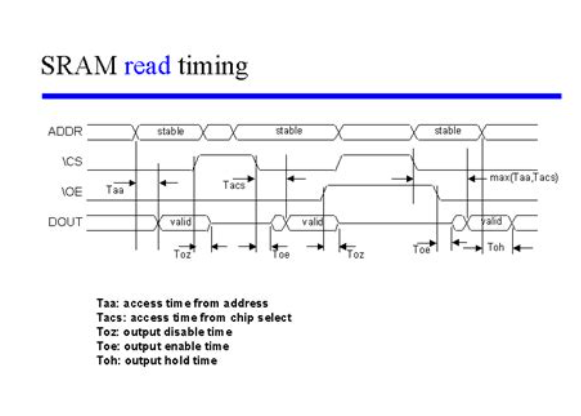
**Write Operation**

When write\_req is asserted for one clock with ready=1, the controller latches address[14:0] and write\_data[15:0] and performs a one‑cycle write. During that cycle it asserts sram\_ce\_n=0 and sram\_we\_n=0 while keeping sram\_oe\_n=1 to prevent the SRAM from driving the bus. The controller actively drives sram\_data with the latched word so the SRAM can sample it on the write strobe. At the following clock edge the controller releases the bus to high‑impedance, deasserts the write control, and reasserts ready.



### **Read Operation**

When read\_req is asserted for one clock with ready=1, the controller latches address[14:0] and performs a one‑cycle read. It asserts sram\_ce\_n=0 and sram\_oe\_n=0 while keeping sram\_we\_n=1. The controller tri‑states its own data drivers so only the SRAM drives sram\_data. The addressed word appears on the bus per the device’s clock‑to‑out specification and is captured into read\_data. The controller then disables the output path and reasserts ready on the next clock edge.



## Data Bus Direction and High‑Impedance

Because sram\_data is shared, exactly one side may drive it at any time. The controller drives during writes; the SRAM drives during reads; between transactions neither side drives the bus, resulting in a deliberate high‑impedance (Z) interval. This avoids contention and provides a clean turnaround between read and write phases.

## Conclusion

The synchronous SRAM controller provides a clean, single‑cycle bridge between system logic and a 32K×16 memory device. By sequencing CE/WE/OE correctly and managing bidirectional bus direction, it abstracts device‑level details into a simple request/ready interface that is easy to integrate and reason about.