TMS320 DSP
DESIGNER'S NOTEBOOK
Number 63
TEXAS INSTRUMENTS

Shared Memory Interface with a TMS320C5x DSP

Contributed by Joe George

Design Problem

How do I efficiently share memory with a DSP and a host without needing expensive dual-port RAMs?

Solution

Various issues should be considered when implementing a shared-memory scheme with the 'C5x. Figure 6-14 and Figure 6-15 in the *'C5x User's Guide* shows the use of HOLD/HOLDA and READY for this purpose. If a no decode interface is desired (see Designer's Notebook page #45), then the SRAMs used should have two chip selects, one for the DSP and the other for the device with which it is sharing memory. In the case of decoded memories, only one select is needed. This means that the accesses must be enabled/disabled from either DSP or the other device, allowing a quasi-dual-port memory scheme sharing one address bus. Figure 1 shows an example of a 'C5x and other processor shared-memory scheme. Even though the 'C5x is usually on the bus, the other processor has priority over the memory. The 'C5x needs to get off the bus before the other processor can access it.

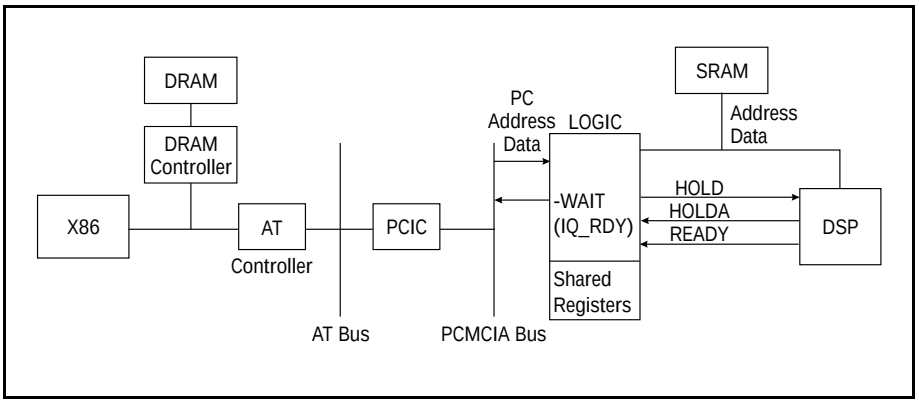

Figure 1. Efficient Shared Memory Example

Bus Arbitration

Bus arbitration schemes can be a difficult problem. In this case, if it is desired to make the other processor have priority over the shared memory, the HOLD/HOLDA and READY operation of the DSP can cause a locked state in a worst-case situation. Thus any arbitration logic should be designed around this and various features could be added to optimize arbitrating operation.

$\overline{\text{HOLD}}$ is the signal that is used by the other device to request the memory bus from the 'C5x. $\overline{\text{HOLDA}}$ is used by the 'C5x to grant the use of the memory bus to the other device. For example, after the bus arbiter asserts $\overline{\text{HOLD}}$, the 'C5x asserts $\overline{\text{HOLDA}}$ in response to the asserted $\overline{\text{HOLD}}$ signal to acknowledge that it is not using the bus. If the 'C5x is on the bus, it completes its current access before it releases the bus and asserts $\overline{\text{HOLDA}}$. One important note is the possibility of $\overline{\text{HOLDA}}$ taking longer than the other device can wait. Thus a "timeout" should be woven into the arbitration logic. $\overline{\text{HOLDA}}$ delay is possible if long wait states are used, especially if the DSP clock has been slowed considerably ($\overline{\text{HOLDA}}$ remains static when clocks are shut off). Also, if the DSP is executing a long RPT instruction, $\overline{\text{HOLDA}}$ will be delayed.

Another issue involves READY signal behavior in the $\overline{\text{HOLD}}$ mode. READY is a 'C5x input signal that is used to extend a memory bus cycle (add hardware wait states). The arbitration logic must supply the READY low signal if the 'C5x external memory transaction cannot be completed in the current cycle. But if the DSP tries to access the shared memory at the same time as the other processor requests the bus and READY has been asserted low by the logic to keep the DSP off, $\overline{\text{HOLDA}}$ may not be returned because the DSP bus cycle has not completed. Thus it is important that READY is not asserted until $\overline{\text{HOLDA}}$ is returned. The disadvantage is that the other processor loses absolute priority over the bus.