

Loosely Coupled Multiprocessing

When your hardware can no longer keep up with the system demands, one solution is to shift to multiprocessing. The idea of improving performance by adding more processors is attractive, even if a moment's thought shows there is more to such designs than adding CPUs and memory. How involved those problems get, however, is often a function of the processor coupling in your design.

In high-end systems, the performance of a closely coupled architecture is clearly superior to that of loosely coupled processors. (For more information on tightly coupled multiprocessing, see Steve Jones's article on the subject.¹) For smaller systems, however, loosely coupled processors can give you a substantial performance improvement without the increased development and product costs of a tightly coupled design. To show how such designs evolve, I'll offer some observations based on two generations of loosely coupled architectures we've used at my company. My particular department develops microprocessor-based blending equipment for the custody transfer of petroleum products.

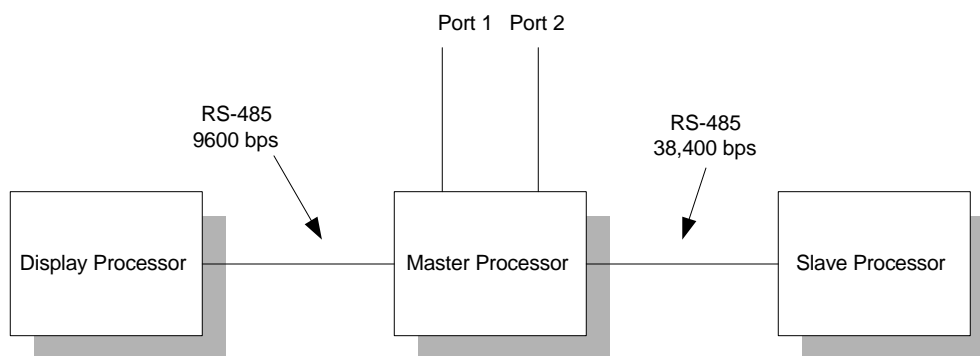


Figure 1 Loosely Coupled architecture

8-bit origins

Our current architecture is given in Figure 1. It consists of two Motorola 68HC11 microprocessors connected in a master/slave configuration through an RS-485 communications link. The master processor is responsible for system synchronization and processing of operator requests. It communicates with a display processor through a standard RS-485 communications link at 9,600 bits per second. Likewise, it employs two serial data communications ports for connection to other products or the customer's host computer. These three ports let the operator or customer interrogate the database, monitor and control the unit, and modify operational parameters.

The master processor receives inputs from the display or communications ports and generates text messages to the display. Request for access to operational parameters are

¹ Jones, Steve. "Tightly Coupled Multiprocessing," *Embedded Systems Programming*, August, 1992, pp. 24-27

channeled through a database manager that determines the processor in which the parameter resides. It generates a local request if the parameter resides in the master processor or an interprocessor message if the parameter resides in the slave processor. The types of messages sent to the slave include how many units of volume to deliver, which product to deliver, and at what flow rate to deliver the product. During the delivery process, the master interrogates the slave for various parameters using a polling technique.

The slave processor is responsible for handling the control valves, pumps, flow meters, and resistive temperature devices. It controls a proprietary bus that connects it to an analog input board and a solid-state relay board. The slave processor manages all delivery process aspects, such as controlling the pumps and valves, processing pulse inputs, maintaining flow rates, and processing temperature data from resistive temperature devices. It also generates pulse outputs that are corrected mathematically to compensate for temperature and pressure effects on the petroleum. It maintains various component totalizers that indicate net and gross quantities. The master polls the slave during the delivery process to determine the delivery status.

Communications between processor boards is controlled by the master. A task in the master requests that a packet be sent to the slave using a kernel call. The kernel returns a mailbox number, which is used by the calling task to interrogate the status of the mailbox. The request is queued by the kernel. The next packet in the queue is sent, and, when the response arrives, the mailbox status is updated, indicating that the data is available. When the calling task determines that the data is available, it reads the data and frees the mailbox.

There are several drawbacks to this technique. First, the processor wastes time polling to see if a response has arrived, and the calling task does no further processing until the response arrives. Another drawback is that the slave can not initiate communications to the master, so a change in the slave processor's status can only be detected by polling, putting an additional burden on the communications link.

In this situation, implementing interprocessor communications uses the 68HC11's serial-communications-interface feature. Typically, packets require approximately 10 milliseconds of transfer time between processors at 38,400 bits per second. Unfortunately, the serial-communications-interface interrupts each time a character is received. Each interrupt typically uses 40 microseconds transferring control to and from the interrupt service routine. For a 48 byte packet requiring 10 milliseconds, 1.9 milliseconds are used in overhead, which is approximately 20%.

Motivation to Change

Due to changes in our market place, which is primarily driven by the actions of the federal government, our company recently had to create a new version of a product as quickly as possible. We investigated tightly coupling the processors and found that we could not afford the development time or the increased product costs. While the existing

architecture had some limitations, it had also proven very reliable in an extremely harsh environment, and we weren't anxious to borrow trouble with a looming deadline. We identified what we felt were the weaknesses in our existing architecture and decided how we were going to improve them.

We knew we needed a new processor with increased address space to support the additional features planned for this product. Our initial search led us to the Motorola 68HC16, however, the fact that the product was fairly new made us nervous and we opted for the more mature 68332. The new processor let us use the queued-serial-peripheral-interface (QSPI) module for the interprocessor communications. The QSPI is a synchronous communications device that allowed us to boost the communications up to 500K bits per second. The QSPI can be programmed to interrupt, whether the queue is empty or full. Since the queue size is 32 bytes, we greatly reduced the number of interprocessor-communications interrupts.

From the software perspective, interprocessor communications is transparent to all tasks in all processors. The interprocessor-communications task on each processor detects any off-board messages and queues them. We defined a packet to be 32 bytes, the length of the QSPI queue. In our application, most messages required only a single packet, although some messages could take as many as seven.

Periodically, the master processor relinquishes control of the link to a slave. Two events can cause the transfer: the queue being empty for 20 milliseconds or the master having sent seven packets. If control is passed to a slave that has no packets, it relinquishes control back to the master, otherwise, that slave can assume mastership of the link. The master will then relinquish control to a second slave, if one is on the link. Transfer of control will occur even if the master is in the midst of a multi-packet message. After the master regains control, it will finish the message.

The Benefits

Since I've gone to the trouble of writing this article, you've probably already guessed that our new design was a success. One of the chief benefits we have seen from the new implementation is much faster interprocessor messaging. Typical interprocessor messages used to require 10 milliseconds to transmit from one processor to another; now it takes 500 microseconds, which is a significant improvement. We also removed the polling requirement, and slave tasks can now initiate messages to the master processor. An additional benefit is that any message transmitted is received by all attached processors, thus, we can have slave-to-slave communications. In our design, however, we don't use that feature, since slave processors are not aware of the existence of other slaves. All in all, we're pleased with the results.

Wayne D. Woodruff is manager of new product development for Brooks Instruments, in Hatfield, PA., a division of Emerson Electric. Woodruff holds a BSEET from Spring Garden College, in Chestnut Hill, PA., and a MEE from Villanova University, in Villanova, PA