Autoloader Firmware Design

When the HP C1553A DDS-2 autoloader was designed, one of the primary goals was to add the autoloader mechanism to a standard DDS drive with no hardware modifications to the drive and minimum firmware modifications. The resulting architecture allows the autoloader mechanism to be independent of drive hardware and to be used with different drive products.

Originally, HP's DDS tape drive product line was not designed to be used with an autoloader. However, the requirement for a low-cost autoloader mechanism was realized and steps were taken to add an autoloader to the current product line. This required three basic steps:

- Design of the drive/autoloader interface, both hardware and software, requiring no hardware changes to the drive
- Addition of the loader command set to the drive firmware
- Design of the autoloader electronics and firmware to enable the required communications with the drive.

Drive/Autoloader Interface

Since the drive had not been designed with the intention of interfacing to an autoloader, there was very little hardware available to create an interface to the autoloader mechanism. It was decided that a port that was used for debugging purposes in manufacturing test could be used to communicate with the autoloader since it had no use outside the manufacturing line.

The port has four data lines and a single address line along with the required handshake lines for the drive's 68000 processor. This allows a total of four registers, two write-only and two read-only. It was decided that these should be 8-bit registers accessed by two successive 4-bit operations. The four registers are the drive status register, the autoloader status register, the drive autoloader command register, and the autoloader drive report register.

Two registers are used as a command report mechanism to allow the drive to send commands to the autoloader. This is the basis for the control of the autoloader. When the drive receives an SCSI command that requires autoloader operation, it writes the appropriate single-byte command code into the drive autoloader command register as two four-bit writes. When the autoloader has completed the operation it places the single-byte report in the autoloader drive report register and asserts an interrupt signal to the drive to indicate that the register should be read.

Commands that require parameters are preceded by a push parameter command. This is a single-byte command that has the top bit set. All other commands have the top bit clear. This allows the remaining seven bits to be pushed onto a parameter stack. Successive push parameter commands allow more than seven bits to be pushed.

The autoloader status and drive status registers are used for handling the front panel. Since the autoloader's front panel is completely different from the standalone drive's, it is accessed via the autoloader processor. However, since the drive processor has the responsibility for telling the autoloader what to do, the front-panel switches are read and interpreted by the drive via the autoloader status register. This allows maximum flexibility of operation and configuration.

To maximize the usability of the autoloader, it was decided to use a character-based LCD display to give messages to the operator. Since most of the status information comes from the drive, the drive status register is used to pass status codes to the autoloader to display status messages on the display. The text for the messages is stored in the autoloader processor ROM. While it would have been more flexible to store the messages in the drive, there was insufficient space.

Drive Firmware Architecture

The changes required to the drive firmware to implement the drive/autoloader interface relate to two distinct areas. The architecture of the firmware for the autoloader is shown in Fig. 1.

First, the normal front-panel handling task within the firmware is replaced by a new version, which communicates drive status to the autoloader. This receives status information from both the SCSI task and the drive task on the drive. This status information is passed over the drive/autoloader interface rather than being displayed on the drive front panel. The SCSI task is changed to read the buttons on the new front panel as well as the eject button on the drive. The drive eject button is left active so that a tape can still be recovered from a drive in an autoloader even if the autoloader hardware is not working.

Secondly, the SCSI task required the addition of the functionality to handle the SCSI medium changer command set. This involved adding new functionality to the task to interpret a new class of commands and pass them on to the autoloader

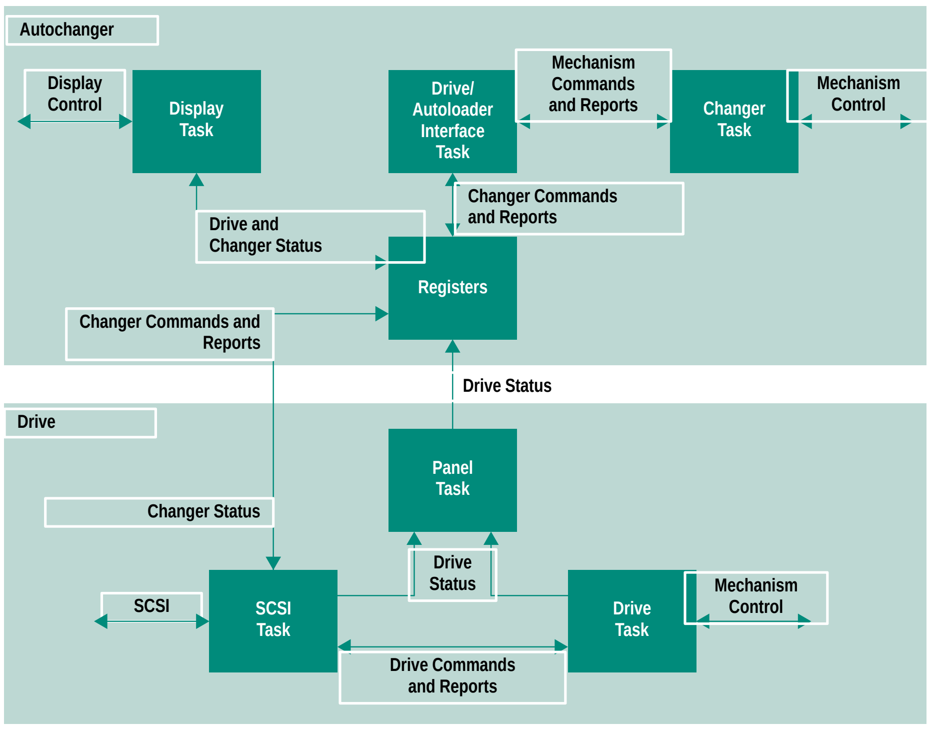

Fig. 1. Autoloader firmware architecture.

mechanism. Parsing the commands on the drive allows the drive to remain in control of the whole autoloader and minimizes the risk of conflicting commands going to the drive and autoloader. It also avoids duplicating the software to parse SCSI commands for both the drive and the autoloader.

Autoloader Firmware Architecture

The autoloader firmware consists of three distinct functions. These are communicating over the interface to the drive, handling the display, and controlling the autoloader mechanism.

These three functions are implemented in three separate tasks running in a round-robin fashion with a 1-ms time slice for each function. This made development of the software easier and allowed the separate functions to be implemented with minimal risk of interference with one another.

To save hardware costs, the drive/autoloader interface registers are not implemented as hardware latches. Instead, the I/O lines from the drive are wired directly into the ports of the H8/325 microcontroller used to control the autoloader mechanism. The H8 has a set of internal memory locations that mirror the imaginary hardware registers. When the select line from the 68000 is asserted, indicating an access to the drive/autoloader interface registers, this causes an interrupt to the H8. The H8 reads its I/O lines and handshakes the data to or from the 68000. This gives the appearance to the 68000 of slow hardware latches. The tasks running on the H8 merely need to access the internal memory locations as if they were the registers.

The drive status register is treated slightly differently from the other registers in the drive/autoloader interface. Because the drive can send repeated status values to this register faster than the display task on the H8 can read them, the values are queued within the H8 to be read in sequence. This ensures that an important status code is not lost behind a less important one. In addition, certain status codes cause flags to be set within the H8 that determine whether the drive is in a certain state. This allows tracking of the state of the drive.

Mark Simms
Development Engineer
Computer Peripherals Bristol