Soft Configuration Enhances Flexibility of Handheld Computer Memory

by Nathan Meyers

SEVERAL IMPROVEMENTS incorporated into the custom CPU of the HP-71B Handheld Computer have considerably enhanced the speed and utility of this processor over those used in earlier HP handheld products. Most notably, a new feature of the bus architecture--soft memory configuration--has greatly increased the flexibility of the system. Soft configuration means that some devices occupying the address space do not have fixed locations; their addresses are assigned to them by the CPU. Putting the responsibility on software for configuring the system allows a tremendous amount of freedom in determining memory layout.

There are two categories of devices that occupy address space: memory (RAM and ROM) and memory-mapped I/O (input/output devices that use a small piece of address space to communicate with the CPU). The HP-71B operating system software is responsible for assigning addresses to all such devices somewhere within the available 512K-byte address space. This task is handled by a 1500-byte routine that executes whenever the computer powers up, cold starts, executes FREE PORT or CLAIM PORT commands or recovers from a module-pulled condition or an INIT sequence.

The challenge in designing the memory configuration software was to come up with a scheme that would offer flexibility in the RAM/ROM mix, maximize the use of the address space, allow for future device types to be defined, and, of course, work within the electrical constraints of the devices being configured.

Memory Layout

The HP-71B's CPU and all other devices reside in a 1M-nibble (a nibble is half of a byte) address space that ranges from hexadecimal address 00000 to hexadecimal address FFFFF. Certain devices are hard-configured as follows:

Address Range	Device(s)
00000 to 1FFFF	Operating system ROMs
2C000 to 2C01F	Card reader interface
2E100 to 2E3FF	Bit-mapped display and timers
2F400 to 2FFFF	RAM in display driver chips (1. 5K)

All other devices are soft-configured in the remaining address space. Specifically, memory-mapped I/O is configured in the space from 20000 to 2C000, extensions to system RAM (memory available to the user) are configured upward from 30000, and ROMs are configured in the space remaining above the end of system RAM (indicated by a pointer called RAMEND).

System RAM begins in the hard-configured RAM (the exact address is a function of system requirements) and extends upward according to how many additional RAMs are available. An HP-71B with nothing plugged into its four ports contains 16K bytes of soft-configurable RAM, so system RAM extends to 37FFF.

System RAM contains a file chain (a linked list of files) and buffer list built upward from low memory, and stacks and variables built downward from high memory, as shown in Fig. 1. Each plug-in ROM contains a file chain that begins + 8 nibbles relative to the beginning of the ROM and is structured exactly like the file chain in system RAM. ROMs contain nothing analogous to the variables, stacks, and buffers stored in system RAM.

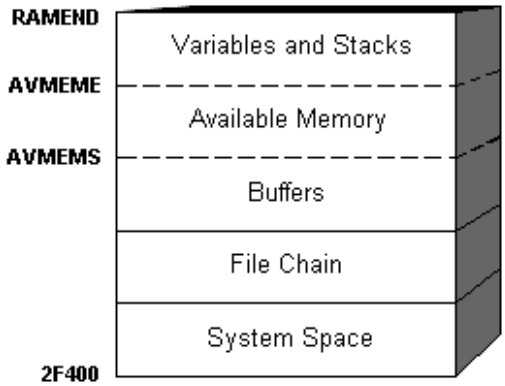

Fig. 1

AVMEME and AVMEME pointers mark the the start and end the of the available RAM address space.

Electrical Behavior of Devices

Soft-configurable devices have data and control lines to interface to the system bus and two additional lines, DAISY-IN and DAISY-OUT, used to make the configuration scheme work. These lines are used in a serial "daisy chain" to establish an order as shown in Fig. 2.

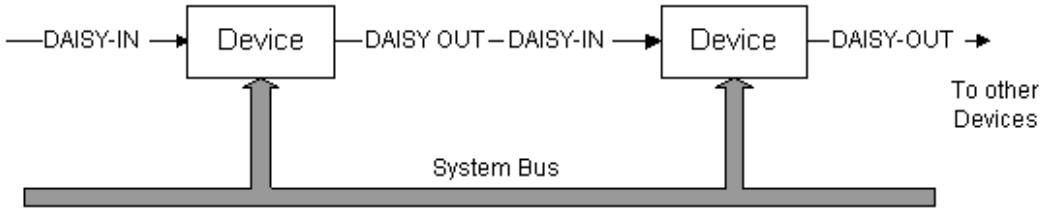

Fig. 2

Devices have two configuration states--configured and unconfigured. When configured, a device occupies address space and responds to normal memory-access instructions (i.e., reads and writes). In this state, DAISY-OUT = DAISY-IN. A configured device can be unconfigured by two bus commands: RESET, which unconfigures all devices, and UNCNFG, which unconfigures a device at a specific address.

When a device is unconfigured, it does not occupy address space. In this state, its DAISY-OUT line is held low. The device will respond to only two bus commands, and only when its DAISY-IN line is high: ID, which causes the device to supply a 5-nibble identification, and CONFIG, which causes the device to become configured to a specified address. These daisy-chain conventions ensure that only the first unconfigured device on the chain responds to these commands. By holding DAISY-IN high to the first device, the software obtains the device's identification, then configures it and moves on to the next device. In this way, all devices on a daisy chain are configured.

The ID command is an important part of the configuration scheme, since it identifies the device's type, size, and other pertinent information. The five digits supplied in response to an ID command contain the following information:

- Nibble 0 = 14 - $\log_2$ (size in K bytes) for memory (value of 9 to F allowed for RAM, 7 to F allowed for other memory) and = 18 - $\log_2$ (size in K bytes) for memory-mapped I/O.
- Nibble 1 is reserved for future use.

- Nibble 2 = device type. 0 = RAM, 1 = ROM, 2 to E = unassigned, and F = memory-mapped I/O.
- Nibble 3 is unassigned if memory. If memory-mapped I/O, 0 identifies the HP-IL mailbox and 1 to F are unassigned.
- Nibble 4's high bit is set if device is last in the daisy chain.

A device cannot be configured to just any arbitrary address. Its configuration address must be on a boundary corresponding to its internal address space. That is, a 1K-byte RAM must be configured on a 1K-byte address boundary. Electrically, the upper nine bits of its address serve as a chip select address while the lower eleven bits serve as internal addressing within its 2K-nibble space.

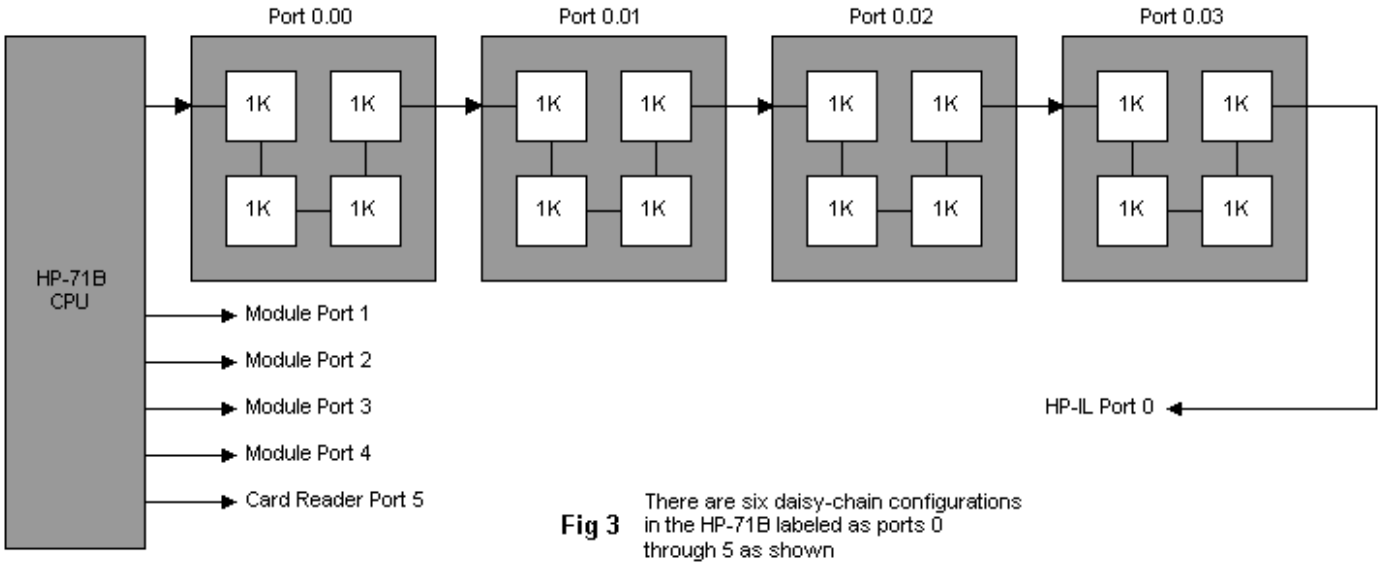

Fig 3 There are six daisy-chain configurations in the HP-71B labeled as ports 0 through 5 as shown

There are six daisy chains, called ports, in the HP-71B (Fig. 3). The first one, port 0, runs through the 16K bytes of built-in system RAM and then to the HP-IL port. DAISY-IN to the first chip in port 0 is tied high. The next four chains are those going to the four module ports in the front of the computer; they are software-switchable lines from the CPU. The last daisy chain, port 5, is a software-switchable line from the CPU to the port for the optional card reader, allowing installation of soft-configurable devices in that port.

Configuration Design Challenges

The configuration software must determine which devices are present, where they should be configured and, most important, how to preserve the integrity of the system when the memory configuration is changed. Some of the major challenges that had to be met during the development of the configuration code are discussed below.

Determining what's out there. The code needs to have a complete inventory of what devices are available before it can decide how to configure the system. To get this information, the software performs an identification prepass, in which it requests each chip's identification and then configures the chip to address 40000. First, all chips in port 0 are identified and configured; this is necessary because the DAISY-IN line to that port cannot be turned off. Once all port 0 chips are configured, the software energizes each successive daisy chain and identifies and configures all chips on that chain. A table is built in hard-configured RAM that will be used to assign addresses later. Configuring all devices to the same address at this time is not a problem as long as the code does not try to access the devices. The chips will later be unconfigured and reconfigured to their assigned addresses.

The table does not contain one entry for every chip. Instead, every sequence of identical chips in a module results in one table entry. A 4K-byte RAM plug-in module, for example, would have a table entry identifying it as containing four 1K-byte chips. An HP-IL module results in two table entries--one for its ROM and one for its electronic interface (mailbox). After the table is built, it is split into three subtables: RAM, ROM, and memory-mapped I/O. Each is handled differently. By processing the chips by device type rather than by port number, the software is fairly insensitive to a plug-in module's exact position. RAMs needn't occupy adjacent ports to work properly. In fact, the port number is not the most important factor in determining a chip's address, as we shall see later.

Extending the user RAM. To be useful, the user RAM must be a contiguous extension of memory from address 30000 (where hard-configured RAM ends). Given the configuration address boundary requirements mentioned above, this presents some difficulty in assigning addresses.

To resolve this difficulty, the RAM subtable is sorted in order of size--largest to smallest. Then addresses are assigned. Since the RAM configuration begins at address 30000 and since all RAM chips must be 512 x 2^n bytes (n = 0, 1, . ., 6), this ensures that all RAM chips can be configured contiguously and on legal address boundaries. (The port number is a secondary sort key. Devices with the same chip size are configured in port number order.) For example, suppose an HP-71B Computer contains the following RAMs:

Port 0:
    Built-in 16K-byte RAM in 1K-byte chips.
Port 1:
    4K-byte RAM module consisting of four 1K-byte chips.
Port 2:
    16K-byte RAM module consisting of four 4K-byte chips. (This is a theoretical example; at this time there is no such module available.)
Port 4:
    16K-byte module consisting of two 8K-byte chips (also theoretical example).

For this case, addresses will be assigned as follows (remember, this is a nibble-oriented address space):

```
30000 to 33FFF: First chip in port 4
34000 to 37FFF: Second chip in port 4
38000 to 39FFF: First chip in port 2
3A000 to 3BFFF: Second chip in port 2
3C000 to 3DFFF: Third chip in port 2
3E000 to 3FFFF: Fourth chip in port 2
40000 to 407FF: First chip in port 0
40800 to 40FFF: Second chip in port 0
. . .
47800 to 47FFF: Sixteenth chip in port 0
48000 to 487FF: First chip in port 1
48800 to 48FFF: Second chip in port 1
49000 to 497FF: Third chip in port 1
49800 to 49FFF: Fourth chip in port 1
```

At this early stage of the configuration code, the chips are not yet configured to these addresses. These addresses are merely written into the RAM subtable. The configuring of chips to their assigned addresses occurs after all chips in all subtables are assigned addresses. For continuity, however, this article will continue to discuss the challenges of system RAM configuration before moving on to other types of devices.

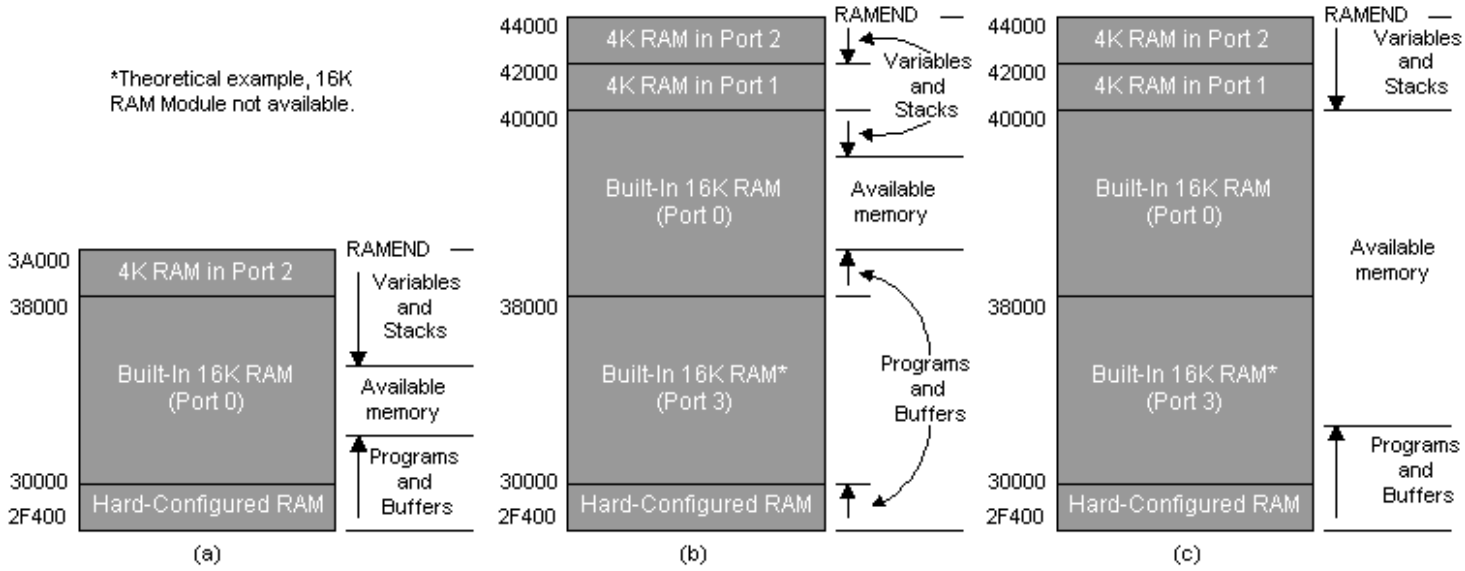

Fig 4. (a) HP-71B system RAM configuration with a 4K-byte RAM module plugged into port 2. (b) System RAM configuration after another 4K-byte RAM module plugged into port 1 and a theoretical 16K-byte RAM module (used for illustration purposes) is inserted into port 3. This leaves gaps or bubbles in the address space which are then moved by the configuration software to the available address space as shown in (c).

Accommodating new RAM plug-ins. What if the user turns off the machine and inserts additional RAM plug-ins? The configuration code must be able to restore system integrity. Therefore, the configuration tables are not thrown away after the system configuration is completed. They are kept in the configuration buffer area in RAM. The tables can then be used to compare the old configuration to the new configuration, and the code can determine what must be done to restore system integrity. For example, assume that an HP-71B is configured with a 4K-byte plug-in memory module in port 2, and that the memory is occupied as shown in Fig. 4a. Now assume that the user turns off the machine and plugs another 4K-byte module into port 1 and a 16K-byte module consisting of two 8K-byte chips (theoretical device used for illustration purposes) in port 3. Following the configuration rules explained above, the memory will now look as shown in Fig. 4b. Adding RAM introduces "bubbles" into the memory that, in this example, disrupt the integrity of both the file chain and the stack space. To solve this problem, the configuration code compares the old and new configuration tables and determines which RAM devices are new (in new table but not in old table) and which RAM devices are missing (in old table but not in new table). If any devices are missing that were not contained entirely in available memory, the computer performs a cold start (memory reset), since this is an unrecoverable disruption of system integrity. If there are any new devices, the configuration code rearranges the contents of memory to restore integrity. By moving data around, it effectively moves the bubbles into available memory (Fig. 4c). The code is general enough to handle any number of new bubbles and restore system integrity properly.

Devising a method to remove RAM modules nondestructively and having a ROM-like RAM, with an independent file chain similar to plug-in ROMs. These two challenges are mentioned together because they have the same solution. That solution is known as "independent RAM."

An independent RAM (IRAM) is treated like a ROM. It is not configured as part of system RAM, and can therefore be removed without destroying system integrity. It contains a file chain just like a ROM, and any IRAM that can retain data when unplugged (a technological possibility) can be used to transfer programs between machines.

The user can designate any RAM to be an IRAM by executing the FREE PORT command, and can unfree an IRAM with the CLAIM PORT command. Since it is not possible for the software to change a chip's identification (to distinguish an IRAM from a RAM), IRAMs are implemented by writing a special sequence of 8 nibbles to the beginning of a RAM module. When the configuration identification code is executed as described earlier, each RAM module is first configured to address 80000 and the first 8 nibbles are read. If they match the special IRAM sequence, the RAM is treated like a ROM; its table entry goes into the ROM subtable and the module is not configured as system RAM.

When the user frees a RAM module, the FREE PORT software verifies that there is enough available memory to remove the RAM. If so, it rearranges memory to exclude the designated RAM, writes the 8-nibble IRAM sequence to it, marks the old configuration table to indicate that this RAM was deliberately removed (so that a cold start will not be performed when the RAM is discovered missing) and executes the configuration software.

This ends the discussion of system RAMs. The problems of configuring ROMs and memory-mapped I/0 are much more straightforward. As with RAMs, the software builds tables identifying which ROMs and memory-mapped I/0 devices are plugged in. The main purpose of these tables is to record where everything is so, for example, the HP-71B's operating system can locate all of the file chains or the 82401A HP-IL Module's software can determine where its mailbox is configured.

Configuring ROMs efficiently in the remaining address space. In all discussion here, "ROM" is shorthand for all memory devices that are not used as system RAM: ROM, IRAM, and any possible future memory types.

Unlike system RAM, ROM has much looser requirements for what goes where. The only restriction is that identical devices in a plug-in be configured together in daisy-chain order. In other words, a 32K-byte ROM module containing two 16K-byte chips must be configured with the chips next to each other in the address space--this keeps the file chain in one piece. There is, however, also a need to use address space efficiently in configuring ROMs. ROMs can get quite large, and 512K bytes of address space is easy to use up. The configuration software handles the problem by sorting the ROM subtable by size (as with RAMs) and configuring from largest to smallest.

Unfortunately, simply configuring upward from RAMEND is usually not possible, since RAMEND will not, in general, occur at a legal configuration address for a large ROM. So the software breaks the ROM configuration into two parts:

1. For large chips (32K bytes and larger), the software configures the module at the highest legal configuration address that will hold it. So a 96K byte ROM module consisting of three 32K-byte chips will be configured at the highest 32K-byte address boundary that is capable of hosting 96K bytes of contiguous memory without configuring over something else (such as the operating system ROM or other large chips).
2. For small chips (16K bytes or less), modules are configured contiguously in the holes remaining after the large chips are configured.

Configuring memory-mapped I/O devices. These are devices that occupy address space, but are not memory. The HP-IL mailbox is an example; it occupies 8 bytes of address space that the CPU reads from or writes to when performing HP-IL operations. To meet this need, a separate part of the address space (20000 to 2BFFF) is designated for use by memory-mapped VO. The sizes handled here are very different from those of memory. Memory-mapped I/O devices typically occupy only 8, 16, or 32 bytes of address space. To make efficient use of the available address space, memory-mapped 1/0 devices are configured in order of their size (as with system RAM).

Configuring everything in daisy-chain order after addresses are assigned. The three subtables are merged and sorted by port number and daisy-chain position. The software then steps through the tables, configuring each device to its assigned address. The subtables are once again separated and saved in the configuration buffer area for future reference.

Acknowledgments

The HP-71B bus architecture, including the feature of soft configuration, was conceived by James P. Dickie and David M. Rabinowitz of HP's Personal Computer Division.

Go back to the HP Journal library

Go back to the main exhibit hall