Running HP 2000 Time-Shared BASIC on SIMH

J. David Bryan, 9-Dec-2017, revised 25-May-2018

The HP 2100 simulator for the 21xx and 1000 series of machines supports execution of the HP 2000 family of Time-Shared BASIC operating systems. The 2000A and 2000E products run on a single CPU, whereas the 2000B, 2000C, 2000F, and 2000 Access products use a dual-CPU configuration. The former present no special problem for execution on SIMH, but the latter require some considerations to run reliably.

The Dual-CPU Hardware Implementation

The HP 2000B, C, F, and Access versions split the TSB operating system into two parts running on separate CPUs. The primary CPU, designated as the *System Processor*, runs the BASIC interpreter for up to 32 concurrent users. The secondary CPU, designated as the *I/O Processor*, handles I/O through the terminal multiplexers. An HP 12875A Processor Interconnect kit provides communication between the SP and the IOP. The kit consists of two bidirectional 16-bit parallel interfaces installed in each CPU. One interface in each CPU is designated as the output interface, and the other is designated as the input interface. Cables cross-connect the interfaces between the CPUs.

To start these TSB systems, a configured IOP program is loaded into the I/O Processor, and the CPU is run. Then a configured SP program is loaded into the System Processor and is run. The system operator interacts briefly with the SP startup routine, which ends with setting the current date and time. The SP and IOP communicate across the processor interconnect, and the system is then ready to accept user logins.

The essential aspect of system startup is that the IOP is running before the SP attempts to communicate with it. If the IOP is not running or is otherwise non-responsive, the SP startup routine halts with the *MUST RELOAD IOP, LOAD ABORTED* error message.

Simulation of the Dual-CPU Configuration

SIMH supports execution of the dual-CPU TSB versions. Two SIMH instances are used—one to run the SP software, and the other to run the IOP software. The *IPL* device simulates the Processor Interconnect kit, with a memory region shared between the two instances serving as the interconnecting cables. SIMH also provides the special IOP firmware that is required to run the 2000 Access version.

Starting the TSB system under simulation has the same requirement as in hardware: the IOP instance must be loaded and running before the SP instance attempts to communicate with it. In addition, the use of shared memory to simulate the CPU interconnection imposes an additional requirement: the memory region to be shared between the SIMH instances must be established before either the IOP or SP begins program execution. These requirements complicate the use of automated simulator startup command files to bring up a TSB system.

Starting a Time-Shared BASIC System on SIMH

To start a dual-CPU TSB system successfully on SIMH, the following sequence of operations must be observed:

- The SP and IOP SIMH instances must be started, and their I/O devices must be configured to match the requirements of the SP and IOP software.
- The shared memory connections between the IPL device of one instance and the IPL device of the other instance must be established.
- A cross-loader program must be loaded into the SP from magnetic or paper tape.
- The binary loader in the IOP must be started to receive the IOP operating software from the SP.
- The cross-loader program in the SP must be run to transfer the IOP operating software from the SP to the IOP.
- When the transfer is complete, the IOP CPU must be run.
- The IOP operating software must complete its initialization work and begin monitoring the processor interconnect for communication requests.
- The SP operating software must be loaded into the SP instance, and the CPU must be run.

In hardware, the first two items above are accomplished when system power is applied. The remaining items are guaranteed by requiring the system operator to load and start the IOP before starting the SP. The physical time required for the operator to attend to the SP guarantees that the IOP initialization routine has completed before SP communication occurs.

Manual configuration of the SP and IOP simulator instances for each execution is tedious and error-prone. A better approach is to have separate SP and IOP command files that configure the CPU and I/O device simulations, set up the shared memory region for the IPL devices, and load the corresponding software. With such files, a direct simulation of the preceding sequence would require the user to:

- start an instance to process the SP command file, then
- start an instance to process the IOP command file, then
- start the IOP binary loader, then
- start the SP cross-loader, then
- start IOP operating software execution, and then
- start SP operating software execution.

The delay inherent in manually starting SP program execution guarantees that the IOP initialization routine has completed before the SP attempts to communicate.

Starting the system in this way is inconvenient. It is preferable to run a single command file that handles all of the subsidiary actions, including cross-loading the IOP operating software and starting the SP and IOP programs. The problem with such an approach is guaranteeing that the startup sequence requirements are met.

Problems with Automatic Startup Command Files

Given SP and IOP command files as outlined above, two options for TSB startup requiring only one user action are possible. One option is to start each instance with its associated command file from a host operating system shell script. The other is to start one instance with its associated command file and have that instance start the second instance with its command file.

As an example of the first approach, a UNIX Bash startup script might contain:

```
xterm -e 'hp2100 iop.sim' &
hp2100 sp.sim
```

The equivalent Windows CMD script would be:

```
start hp2100 iop.sim
hp2100 sp.sim
```

A problem with this approach is that the two instances execute their command files asynchronously, so the requirement that the IOP starts first cannot be guaranteed.

For the second approach, the user starts the first SIMH instance manually, specifying the SP command file:

```
hp2100 sp.sim
```

...and at some point within the SP file, there is a SIMH spawn command:

```
! start hp2100 iop.sim
```

...that starts a second instance of the simulator to execute the IOP command file. This approach eliminates the need for the host system shell script and guarantees that commands in the SP file preceding the spawn of the IOP instance are executed before the IOP commands. However, this still does not guarantee that the IOP starts first or completes its initialization before the SP program attempts communication. If the host operating system blocks the IOP instance from executing due to higher priority contending processes, the SP command file will continue through SP program loading and execution, which will fail when IOP communication is attempted.

The problem is exacerbated with the advent of multi-core PCs. With a single-core host machine, the SP and IOP instances must execute alternately. When the IOP instance is started, it may receive enough CPU time to load and initialize before it is preempted and the SP resumes execution. Processor contention, either from the host OS itself or from other programs executing concurrently, must suspend both SIMH instances. This improves the probability that the IOP will complete before the SP.

With multi-core machines, the SP may execute either concurrently with the IOP or while the IOP is blocked by processor contention. A complicating factor is that SIMH automatically decreases its own process priority when simulated execution begins with a RUN or GO command. Consequently, an instance executing SCP commands has priority over an instance executing TSB system code. Successful system startup then becomes much more sensitive to the timing of operations within the respective SIMH command files.

There is no general solution to these problems because there is no way to guarantee that the SP and IOP instances will not be preempted by the host operating system. In hardware, the IOP takes a deterministic time to execute its initialization code. In simulation, this time will vary from some minimum lower limit to essentially an unbounded upper limit, depending on host system load. The best approach is to increase the probability that the IOP completes before the SP and therefore that the system startup will succeed.

Improving Startup Probability with Process Synchronization

While the required sequence of startup operations cannot be guaranteed with independent SP and IOP instances, the probability of following the proper sequence may be improved considerably by synchronizing the processes at certain critical points. Essentially, this involves suspending one process until the other process "catches up" to the same point in the startup sequence. Release 28 of the HP 2100 simulator added two SCP commands for this purpose. The ***SET IPL WAIT*** command suspends a simulator process until a matching ***SET IPL SIGNAL*** command is issued by the other process. If the signal command is issued before the wait command, the latter will not suspend but instead will proceed immediately with the next command.

As an example, consider the 2000 Access cross-loading sequence that loads the IOP operating software. In hardware, the required sequence is:

- Mount the IOP tape on the magnetic tape drive.
- Enable and run the resident magnetic tape bootstrap on the SP to load the cross-loader into memory.
- When the bootstrap halts, set the starting address and run the cross-loader on the SP.
- Wait for the message *START IOP PROTECTED LOADER. PRESS RETURN* to be printed on the SP system console.
- Enable and run the resident processor interconnect bootstrap on the IOP to receive the IOP program. The IOP is now waiting for the SP to transmit.

- Press the RETURN key on the SP console to begin transmission.
- The IOP halts when the reception is complete.
- The SP halts when the transmission is complete.

The critical requirement is that the processor interconnect bootstrap must be running on the IOP before RETURN is pressed on the system console to begin SP transmission. The bootstrap performs a hardware clear on the processor interconnect interface before waiting for the first word to arrive. If the SP has already transmitted the first word, the hardware clear will remove it, and the cross-load will be corrupt.

To avoid this in simulation, process synchronization may be used suspend the SP process until the IOP processor interconnect bootstrap has started. An SP command file that provides the required startup sequence might contain these lines:

```
attach -E MSC0 IOP.tape
boot MSC0

assert T=102077

deposit P 002000
go until "START IOP PROTECTED LOADER.  PRESS RETURN"

! start hp2100 iop.sim

echo Waiting for the IOP to indicate that it is ready to receive...

set IPL WAIT

echo Starting the IOP program transfer.

send "\r"
go

assert T=102077
```

...where the ***start*** command above is used to spawn an asynchronous HP 2100 instance, and the ***assert*** commands are used to ensure that the bootstrap and cross-load transfers completed successfully with HLT 77 instructions. The corresponding IOP command file might contain:

```
echo Starting the IOP binary loader.

deposit S 000000
load IPL
step 20

echo Signaling the SP that that the IOP is ready to receive its program.

set IPL SIGNAL

echo Receiving the IOP program.
```

```
go

assert T=102077
```

The ***load IPL*** command loads the processor interconnect bootstrap into memory, and the ***step 20*** command executes the bootstrap through the hardware clear and into the reception loop. At that point, it is safe for the SP to begin sending the IOP program across the interconnection. The ***set IPL SIGNAL*** command releases the SP program, which has been waiting at its ***set IPL WAIT*** command, to press the RETURN key and begin the cross-loading process.

After cross-loading is complete, the message *I/O PROCESSOR MAY BE STARTED AT LOCATION 2* is printed on the SP system console. The system operator then sets the starting address and runs the IOP before returning to the SP and initiating the disc bootstrap.

In simulation, a synchronization point is needed to wait for IOP initialization to complete before booting the disc. This requires a ***set IPL WAIT*** command in the SP command file before the ***boot*** command. In the IOP command file, the cross-loaded program is run until it reaches the address of the end of IOP initialization routine. At that point, a ***set IPL SIGNAL*** command releases the SP to proceed with the disc bootstrap.

It is important to understand that process synchronization substantially improves, but does not guarantee, the probability of a successful startup. TSB subjects SP-to-IOP communications to timeouts that will result in the *MUST RELOAD IOP, LOAD ABORTED* error message if they expire. Such a timeout might occur if the IOP process is preempted for a long time immediately after a ***set IPL SIGNAL*** command releases the SP process to continue.

In addition, some host platforms do not support the underlying system library routines necessary to provide process synchronization. In this case, the ***set IPL WAIT*** command falls back to a simple two-second timed wait, which may provide enough host processor time for the other process to reach its ***set IPL SIGNAL*** command (which, in fallback mode, has no effect). If this is insufficient, explicit SCP ***sleep*** commands may be added to lengthen the pause.

The IPL device writes a trace line when an ***attach*** command is issued and the host system does not support process synchronization. For example:

```
set debug stdout
set ipl debug=cmd

attach -S IPL 1
>>IPLI cmd: Synchronization is unsupported on this system; using fallback
```

The Effect of Throttling on System Startup

HP 2000 Time-Shared BASIC does not have a traditional localized idle loop that is executed while the operating system is otherwise unoccupied. Consequently, the SIMH

set CPU IDLE command does not idle the simulator when running TSB, and therefore the SP and IOP instances take 100% of their available host CPU times.

The SP and IOP command files may contain ***set throttle*** commands to reduce the loads on the host CPU. However, because throttling periodically preempts the SIMH process, achieving a successful system startup in the presence of throttling is even more problematic than it is otherwise. If throttling is used, startup probability may be increased by enabling throttling only after IOP initialization is complete.

The Race Condition in 2000 Access

The 2000 Access version has a race condition that manifests itself by an apparently normal boot and operational system console but no ***PLEASE LOG IN*** response to terminals connected to the multiplexer. The frequency of occurrence is higher on multi-core host systems, where the SP and IOP instances may execute concurrently.

The cause is this code in the SP disc loader (source files S2883, S5ISS, S7900, and S7905 on the Access source tape):

```
LDA SDVTR      REQUEST
JSB IOPMA,I      DEVICE TABLE
[...]
STC DMAHS,C    TURN ON DMA
SFS DMAHS      WAIT FOR
JMP *-1          DEVICE TABLE
STC CH2,C      SET CORRECT
CLC CH2          FLAG DIRECTION
```

The STC/CLC at the end normally would cause a second "request device table" command to be recognized by the IOP, except that the IOP DMA setup routine *DMAXF* (in source file SD61) has specified an end-of-block CLC that holds off the IPL interrupt, and the interrupt completion routine *DMCMP* ends with a STC,C that clears the IPL flag.

In hardware, the two CPUs are essentially interlocked by the DMA transfer, and the DMA completion interrupts occur almost simultaneously. Therefore, the STC/CLC in the SP is guaranteed to occur before the STC,C in the IOP. Under simulation, and especially on multi-core hosts, that guarantee does not hold. If the STC/CLC occurs after the STC,C, then the IOP starts a second device table DMA transfer, which the SP is not expecting. Consequently, the IOP never processes the subsequent "start timesharing" command, and the multiplexer does not respond to user login requests.

The simulator employs a workaround that decreases the incidence of the problem: the DMA output completion interrupt is delayed to allow the other SIMH instance a chance to process its own DMA input completion interrupt first. This improves the race condition by delaying the IOP until the SP has a chance to receive the last word, recognize its own DMA input completion, drop out of the SFS loop, and execute the STC/CLC. The delay is initially set to one millisecond but is exposed via a hidden IPLI register, *EDTDELAY*, that allows the user to lengthen the delay if necessary.

The condition is only improved but not solved because delaying the IOP does not guarantee that the SP will actually execute. It is possible that a higher-priority host process will preempt the SP, and that at the delay expiration, the SP still has not executed the STC/CLC. Still, in testing, the incidence dropped dramatically, so the problem is much less intrusive.

Summary

The HP 2000B, C, F, and Access dual-CPU Time-Shared BASIC operating systems will run on the HP 2100 simulator, but several internal aspects must be considered for successful startup and operation. These arise from the differences in behavior of dedicated hardware versus two simulator instances running on a multi-core host machine. Arbitrary host OS preemption of the separate SP and IOP instances means that the deterministic behavior of the original HP hardware cannot be guaranteed. However, the use of process synchronization commands during system startup will improve the probability of successful system execution.