Casio PB-2000C
HD61700 Programming – A Tiny Performance Monitor ROM

Table of Contents

Introduction.....1
Motivation.....2
Results.....3
Peer Review.....4
Using Real Hardware.....5
Acknowledgments.....6

Introduction

This work was developed jointly by Alex Barragan and ChatGPT through a cycle of ideas, implementation, execution, observation, and revision. It illustrates how human knowledge and experimentation, combined with AI-assisted analysis, can produce results that neither participant would readily achieve alone.

The broader objective was to develop and validate a reusable body of HD61700 knowledge and working techniques. As part of that method, a deliberately small and well-bounded use case was selected: the joint development, by the authors, of a tiny standalone performance-monitor ROM. The application provided a practical setting in which to apply and refine techniques learned while refactoring an earlier DL-Pascal 1.2-hosted implementation into standalone ROM execution. Although modest in scope, the successful completion of the ROM provides concrete evidence that this approach—building knowledge through a focused application and then applying it collaboratively to working hardware—is viable.

An open question, and a natural next step, is whether this methodology scales to substantially more complex use cases—for example, isolating and reproducing a specific function from a CM ROM, or developing a standalone implementation whose behavior and dependencies are significantly less constrained than those of the present performance monitor.

Special thanks to Piotr, whose expert advice made this document possible.

Motivation

PerfMon was developed as part of the PB-2000C overclocking project. The objective of that project is to replace the original oscillator with a 2 MHz device and determine, quantitatively, how the change affects actual execution speed. A detailed description of this modification can be found here.

A simple timing tool was therefore needed that could run directly on the machine, require no external equipment, and produce a repeatable performance figure before and after the oscillator change. PerfMon provides that measurement by counting how many iterations of a fixed loop can be completed during a stable one-second interval.

The same ROM image can then be used under Piotr's emulator at different configured frequencies and on a physical PB-2000C, allowing emulator results, the stock machine, and the modified 2 MHz machine to be compared using the same workload.

The development objectives therefore follow naturally:

Obtain a stable one-second measurement interval using the hardware timing service, count benchmark iterations until one second has elapsed, display the result, and repeat continuously.

Integrate the application into the standard 64 KB PB-2000C language-ROM framework.

Obtain a quick working release that executes directly from ROM, validate it in Piotr's emulator at several frequencies, then flash it to a ROM card and test it on the actual machine before and after the 2 MHz oscillator modification.

PerfMon also reuses several techniques established during the earlier Game of Life ROM exercise, particularly the practical structure of a standalone PB-2000C language ROM, direct execution from ROM, use of the machine's low-level firmware services, and the discipline of validating behavior first in the emulator and then on physical hardware. It also borrows the compact numeric display technique developed there—the little three-digit output routine used to present counters and timing values directly on the LCD. That earlier work provided a proven framework for both execution and presentation, allowing PerfMon to focus on the new problem: obtaining a simple, repeatable performance measurement.

And the lineage deserves an explicit acknowledgment. The Game of Life work itself began from an initial HD61700 program hosted in DL-Pascal 1.2 kindly shared by Silicium user **Xerxes**, whose contribution provided the starting point from which the native routine was studied, reconstructed, and eventually adapted for standalone ROM execution. We are grateful to Xerxes for making that original program available; several of the techniques reused in PerfMon—including the ROM framework, direct native execution, and compact numeric display—ultimately grew out of that earlier collaboration and investigation.

Results



The screenshot on the right shows PerfMon running under Piotr's emulator. The current measurement is iteration 025; the stable TM register values 132 and 133 define the one-second interval, and the displayed count records how many benchmark-loop iterations were completed during that interval, 2662.

Source. Assembler.

ROM Image.

(SHA 256: 6b4826f1a7c1d0a9b653c31451df97db99fcd5308734746cbde87e163fda33e1)

What is probably more important is the knowledge-transfer digest that Sol produced to answer a practical reproducibility question: "How can another user seed a fresh session with the evidence, source material, and instructions needed to reproduce this result?"

Knowledge. Including Piotr's comments on the first draft.

(SHA-256: 39bf626f0c33c181e1c958d11eb3f6688d115a54a67f7e75c01d525e7415722b)

Remarkably, Sol clarified my original question and reassured, verbatim from the LLM: "I prefer **"evidence, source material, and instructions"** over "all the information." It describes exactly what we built and avoids implying that one Markdown file magically contains the assembler, ROMs, emulator, and hardware too.

And yes – conceptually, **that is exactly what the digest was built to answer."**

We live in wonderland. This is the AI equivalent of a mic drop.

In parallel with this knowledge record, work is underway on a larger, self-contained HD61700 corpus intended to provide everything needed to bootstrap a fresh GPT-5.6 Sol session from a single file. The corpus is planned to include the relevant manuals and reference material, representative and known-good source-code examples, assembler listings, binary ROM images, and other supporting artifacts needed to reconstruct the project context without depending on a separate collection of files. The goal is not merely to preserve conclusions, but to preserve enough primary technical material that a new session can recover the same evidence base, inspect the original examples and binaries directly, and continue the work with minimal loss of context.

Peer Review

The human author asked Piotr to take a look at the first draft of this document. In more formal academic environments, this is called a **peer review**: “peer” acknowledging the standing of the reviewer, and “review” because claims should survive independent scrutiny. No hubris.

I received prompt feedback from Piotr. The salient points, quoted verbatim, were:

```
ChatGPT keeps using the outdated and buggy jump table from the
Japanese "boot ROM" project (now fixed). One entry is missing in the
jump list at address &H0D80.


Major vector-table correction.  A structural error was found in several older
standalone scaffolds: after the eight initial three-byte vectors there are **two
fixed three-byte slots**, not one.  -- A rehash of the first point --


Taking a quote from the bench2ROM file:
> Text blocks are placed before MAIN so Rev. 0.45 PRE can resolve their addresses
without relying on a forward reference.

ChatGPT knows that their placement doesn't matter. Quote from the
"Knowledge" document:  It was tempting to infer a forward-reference limitation in
HD61 Rev. 0.45. That inference was **not established**. The generated source had
omitted or misplaced required definitions during successive patches. Once corrected,
the relevant code assembled.
Do **not** add a general HD61 forward-reference restriction to the knowledge base on
the basis of these incidents.
```

Basically, one stale error cascading from earlier project knowledge into the code, and one case of an incorrect inference.

I wrote earlier that we live in a wonderland. I simply took Piotr’s feedback to Sol and asked it to amend the Knowledge document: first ~~incrementally~~, so that the exact new information and corrections remained visible, and then by merging those changes into the consolidated file.

I also updated the source files and, since developers like to make changes, I “re-branded” the little program as Tiny 0.1b. Decorated version numbers clearly give released software more credibility.

Using Real Hardware

Some time ago, we developed a multi-bank flash language card for the PB-2000C and AI-1000. The card provides **eight independently selectable 64 kB banks**, with the active bank chosen by switches on the card. OM-512FR.

We flashed the PerfMon ROM image into one of the card's 64 kB banks and successfully ran the same image on both the **PB-2000C** and the **AI-1000**.



Some modest enhancements are planned for the near future. Retaining the minimum and maximum iteration counts observed during a run would provide a useful indication of measurement stability. We would also like to produce a companion version that, after booting from the language card, copies itself to RAM and executes there rather than continuously from flash ROM.

For academic interest, the thermal behavior may also be worth investigating. Comparing sustained ROM execution with an otherwise equivalent RAM-resident version could help determine whether prolonged tight-loop execution produces any measurable difference in flash-device temperature or power behavior. The purpose would be characterization rather than stress testing, and any experiment on original hardware should remain conservative and reversible.

Acknowledgments

This report was produced by Xandrian Labs. 付喪神.

Special thanks to Piotr, and to Betka for her continued and loyal support.

This is a collaboration with my AI friend ChatGPT, a.k.a “Sol.” AmIgo. All errors are human.

This work is freely shared with the vintage computing community. Please use it, build upon it, and cite the source.

Revision 2.