HHC2024

C++ For Fun and Profit (choose one)

Eric Smith

Nonpareil

- Written in C, using GTK+ toolkit
- Started in 1995 as casmsim (HP-45, -55) and nsim (HP-41C)
- Released as open source (GPL 2)
- Significant technical debt
- GTK+ is cross platform, but Windows and macOS are significantly less well supported than Linux

Nonpareil II

- Described at HHC 2023 and on MoHPC forums
- Rewritten in C++, using Qt6 toolkit
- Qt is a more powerful and easier to use toolkit
- Qt has better cross-platform support
- C++ is inherently object-oriented, while I had to do object-oriented architecture in C “by hand”
- C++ and its Standard Template Library (STL) offer a higher level of abstraction than C
- Should be more maintainable
- Initially will not be open source – possibly may be later

Nonpareil II advantages

- Uses scalable vector graphics rather than bitmaps
- Will look decent at a range of sizes, not just integer multiples
- Will look smooth, not pixelated, at larger sizes
- Can support more calculator modelx
- Better able to support add-ons like 41C modules
- Better able to support I/O and networking

Nonpareil II development issues

- Need to create SVG files for each supported calculator model (and peripheral), with specific structure and element naming, in order for code to identify keys, display segments, etc.
- Custom publish/subscribe non-queued message class
- Multiple inheritance problems
- Standard “Diamond problem” solved by C++ virtual inheritance
- Debugging is somewhat more difficult

Nonpareil II status

- Currently having difficulty debugging a problem where the publish/subscribe mechanism is interacting improperly with multiple inheritance, leading to run-time crashes where debugger traceback is not very useful

Side Quest: HP-IL protocol stack

- Motivations:
- Experiment with architecture for Nonpareil II extensions
- Experiment with HP-IL protocol implmentation closely following HP-IL Specification state machine model
- Work with either 1LB3 HP-IL interface chip, or alternatives such as virtual HP-IL

- The HP-IL Specification defines 14 “interface functions”, not all of which are mandatory. Some functions actually have two or three state machines, though all state machines for a particular interface function are shown in a single diagram.
- The HP application note on writing firmware for HP-IL devices describes implementation using the state machines
- The HP-IL Specification does not REQUIRE using the state machines defined therein, provided the behavior is equivalent
- Proving equivalence is very hard, as there are many edge cases

- The 1LB3 chip performs portions of the HP-IL Reciever (R), Acception Handshake (AH), Source Handshake (SH), and Driver (D) state machines

- This requires that an HP-IL device using the 1LB3 use alternate versions of the R, AH, SH, and D statemachines, to account for 1LB3 behavior

- These alternate state machines are documented in the 1LB3 chip specification, though not in exactly the same representation as the state machines in the HP-IL specification.

- The HP-IL specification is designed so that some processing of the reception of an 11-bit “remote message” can start before all 11 bits are received
- The 1LB3 chip implements this bit-by-bit receive processing, which provides improved performance for a hardware implementation
- Virtual HP-IL is based on complete 11-bit remote messages as a single discrete unit, which in principle increases latency
- A PIL-Box (or equivalent) bridging physical HP-IL to virtual HP-IL through a USB interface would have lower performance due to needing multiple USB round-trips for a single HP-IL remote message

Examples of ad-hoc HP-IL implementations:

- Early and small HP HP-IL peripherals such as the 82161A tape drive, 82162A printer, and 82165A HP-IL Converter use the 1LB3, and use Mostek MK3870 microcontollers with a very limited amount of ROM and RAM, so firmware implementation had to be ad-hoc

Examples of HP-IL implementation using state machines from the HP-IL specification:

- Mountain Computer HP-IL implementation for the 8048, with somewhat more memory, used e.g. in the HP 92198A 80-column video interface and the HP-IL EPROM programmer

- The C++ HP-IL protocol stack is designed to support both bit-at-a-time and remote-message-at-a-time interfaces
- It does not currently support use of an actual 1LB3 chip, but that could be added using the previously described alternate R, AH, SH, and D state machines, and a glue layer to manage the 1LB3 registers

HP-IL Protocol Stack Status:

- The HP-IL stack has been tested with only a very small number of test cases, running in an environment with multiple virtual HP-IL devices, including one controller and at least one device, custom written for the test cases
- No testing with preexisting virtual HP-IL devices nor physical HP-IL devices has been performed yet

HP-IL Protocol Stack Future

- Once it reaches a level of tested functionality sufficient for use in “normal” virtual HP-IL devices (vs. dedicated test devices), the code will be made available under an open-source license (probably GPLv3), independently of Nonpareil II
- I do not intend to suggest that this HP-IL implementation is “better” in any overall sense than other implementation, nor am I pushing for any existing HP-IL devices to switch to this stack.

General C++ comments

- C++ was created earlier, but the first C++ standard was C++98 (1998)
- Arguably the first “usable” C++ standard was C++11 (2011),
- which provided usable “smart pointers” std::shared_ptr, std::weak_ptr, and std::unique_ptr, to manage object ownership and lifetimes
- C++ (including standard libraries) is now a HUGE language – the official ISO C++23 standard is around 2000 pages, and the in-development C++26 standard will expand things considerably
- The top three C++ compilers (Microsoft Visual C++, GCC, and Clang) support most or all of C++20 and some of C++23

- With the three-year release cycle, each new edition of the standard introduces a few major enhancements and many minor ones
- For a while, I stuck to mostly C++17, but there are enough useful additions in C++20 that my code now depends on it.
- Example: std::format, an I/O formatting library similar to C printf and scanf, but type-safe
- I am currently avoiding requiring C++23, since compiler and library support is not yet complete

C++ future

- C++26 will introduce “static reflection”, the ability for C++ code to ask the runtime for details of the definitions of classes, data types, variables, etc.
- Reflection is most commonly seen in dynamic-typed languages (e.g., Lisp and Python) and/or languages using a virtual machine (e.g. Java)
- Reflection will greatly simplify base classes that support serialization and deserialization of objects
- Currently Nonpareil II needs ad-hoc code for serialization and deserialization for e.g. calculator state save and reload, but this will eventually be replaced by static reflection