Designing the Next-Generation Handy Board

Fred G. Martin and George J. Pantazopoulos

University of Massachusetts Lowell
Computer Science Department
1 University Avenue, Lowell, MA 01854
fredm@cs.uml.edu, gpantazo@cs.uml.edu

Abstract

The Handy Board is a robot controller board widely used in undergraduate education. It has a feature set optimized for mobile robotics in the classroom, including Interactive C, an easy-to-use development environment. This paper analyzes the features that made the Handy Board successful, provides a framework for understanding tradeoffs in the computational power of embedded systems and the development environments that can be used with them, and presents plans for a next-generation Handy Board design.

Introduction

The Handy Board is a palm-sized robot control board that was developed in the early 1990s for use with the undergraduate "6.270" LEGO Robot Design Competition at the Massachusetts Institute of Technology (MIT). The full hardware design (including PCB artwork) and a version of its specialized "Interactive C" software were released to the public with an open-source license in 1995. The design is now in widespread use worldwide, in universities, colleges, and secondary schools. Here, we describe the qualities that made the Handy Board design successful, and present goals and initial plans for a next-generation design. This paper is meant as a catalyst for conversation; we welcome input regarding our work plans.

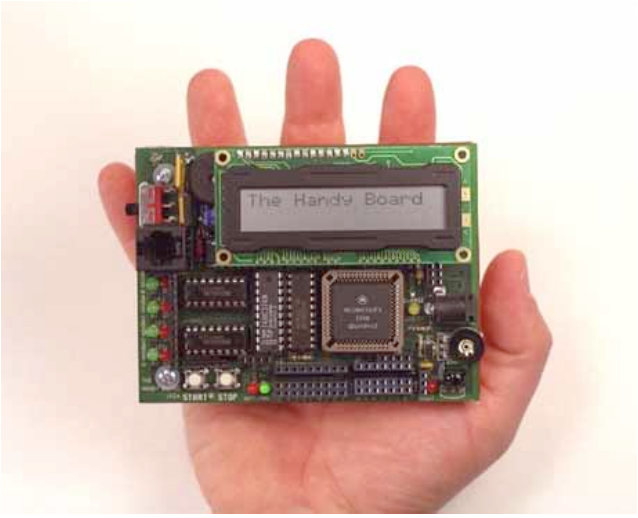

Figure 1: The Handy Board

Why the Handy Board is Successful

The Handy Board (Figure 1) is a remarkably durable design, with essentially no design changes in 10 years, and only minor updates from an earlier version used in the 1991 MIT robotics course (Martin, 1994). It's worth considering why this is so.

Most importantly, the Handy Board was designed for a very specific target application: building small mobile robots in a classroom environment. It includes a comprehensive feature set for this application: on-board DC motor drivers, individual connectors for sensors, a built-in rechargeable battery pack, an LCD screen for status messages, non-volatile program memory, user buttons, and a beeper.

The Handy Board comes with Interactive C (IC), an innovative C-language software system for embedded programming (Sargent, 2004). IC is a combination of compiler that runs on a desktop/laptop workstation PC, and a stack-based virtual machine (VM) that runs on the Handy Board. The IC compiler outputs code for the custom stack language interpreted by the VM. IC includes an interactive, Lisp-like console that lets the user dynamically type individual statements and code blocks for immediate compilation and execution on the Handy Board. This gives the system the flavor of a fully interpreted language. Interactive C "just works," and makes embedded programming accessible to novices. With the recent support of the Kiss Institute for Practical Robotics, the IC application has been updated for today's latest desktop operating systems.

Other factors include:

A fully documented, open hardware design. The Handy Board's hardware design is documented on the web and in print (Martin, 2001) and has an open-source license. The documentation encourages users to understand how the design works and adapt it to their needs, be they pedagogical or practical. The open-source license has encouraged adoption in terms of community good will and also has facilitated its use in developing countries (printed circuit boards can be locally fabricated with no licensing fees).

Sturdy and easy to repair. In normal classroom use, the board is routinely abused. The design uses "high-speed" CMOS chips (introduced in 1982), which are fairly robust. When repairs are needed, all but one of the chips are socketed, allowing easy replacement.

Relatively low cost. In the United States, the Handy Board retails for $300, which is a good value considering its included features.

Limitations of the Current Design

The primary limitation of the Handy Board is its raw computational horsepower. It uses an 8-bit CPU running with a 2 MHz system clock (the Motorola 68HC11) and has only 32k bytes of main memory. For simple robot programs of a few pages or so of code this is adequate, but once one desires to do something more complicated (like mapping or behavior-based robot programming) the Handy Board will come up short.

Along other dimensions, such as sensor and actuator I/O, the Handy Board does quite nicely. The main board has outputs for 4 motors and inputs for 16 sensors (9 digital and 7 analog). When equipped with its Expansion Board, it gains 6 servo motor outputs, 8 digital outputs, and 14 additional analog sensor inputs.

Alternatives to the Handy Board

The Handy Board has at least a few significant competitors:

The LEGO RCX Brick. As part of the LEGO Mindstorms Robotics Invention System launched in 1998, the LEGO RCX controller was an immediate hit, especially with adults. For the first time, a person with no prior electronics or computer expertise could build a fully functioning autonomous robot. More recently, advanced software environments that work with the RCX brick have been developed (including support for Common Lisp, Java, and machine learning algorithms), allowing it to be used with undergraduate computer science majors (Klassner and Anderson, 2002).

The advantages of the RCX Brick over the Handy Board are lower cost ($200 for the RCX Brick including sensors, motors, and the LEGO elements required to build a small robot), and better connectors.

The BASIC Stamp. For those more interested in electrical engineering than coding, the BASIC Stamp is a good option (Parallax, 2004). First released in 1992, the BASIC Stamp is now a family of devices that combine small microprocessors with a BASIC virtual machine and an easy-to-use desktop programming environment. The BASIC Stamp processor itself sells for as little as $29. A full project requires that users add their own breadboard, wiring, sensor circuits, power supply, etc—in other words, to do circuit design. This makes it somewhat less desirable if the goal is to quickly build and then program a mobile robot.

The Stamp has been a success because of its software; its BASIC language is easy to learn, and the software and documentation let users get up and running quickly. Also, the device comes with a nice set of signal-level drivers, such as servo motor outputs and serial communications.

Single-Board PCs and Handhelds. Another approach is to embed an entire PC into the robot. The industry standard "PC104" is a small form-factor PC-on-a-board, with support from multiple vendors. (These systems are surprisingly expensive—well more than low-end desktop PCs.) Also, one can use a Palm or PocketPC device as a robot's brain, and have it communicate with an external sensor/actuator IO board.

Design Goals for the Next Generation

Above all, a new Handy Board should have the same ease of use that characterizes the current design. It should "just work." In framing the overall design objectives, we will keep in mind the same application that motivated us in the past: the development of a controller of a small classroom robot. Based on the previous discussion, the main thing we need to add is CPU "zorch"—the existing Handy Board has a reasonable complement of device I/O.

Let's look more closely at what kinds of software scenarios would be possible with a faster CPU and more memory.

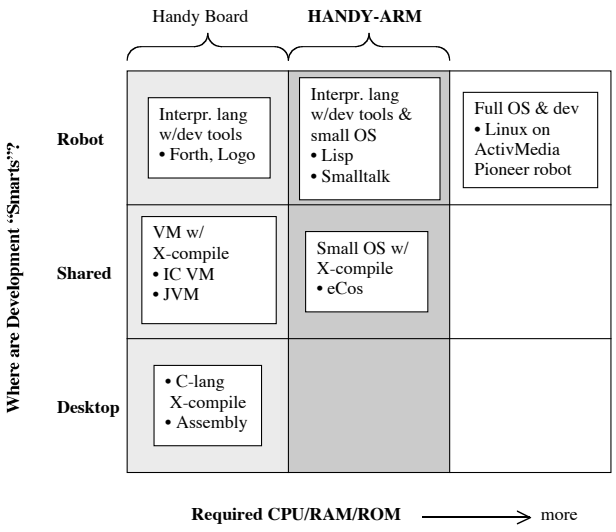

Figure 2: Location of Development Smarts Versus CPU Power

In Figure 2, the vertical axis indicates the locus of software development "smarts." The bottom row indicates software running on a desktop workstation; e.g., a compiler or an

assembler. The top row indicates software running on the robot controller board; e.g., an interpreted language, or an operating system.

The horizontal axis shows how much computing horsepower is required to accomplish a particular software development environment. On the left are low-powered systems; on the right are high-powered ones.

Now let's look at specific points in this space. In the lower-left corner is the traditional embedded development environment. A desktop computer runs a cross-compiler ("X-compiler" in the diagram) or an assembler, and the resulting code is loaded onto the target system. With this development framework, while target may be anything, from the smallest chip to a powerful computer, this is the least interactive, least interesting point in the space.

Moving up the left column is one of the "shared" options, where development smarts are located both on the desktop and the robot. This position is represented by the virtual machine (VM) approach. A VM runs on the robot platform, and the desktop runs a cross-compiler for this VM. Interactive C running on the Handy Board is in this space; so are the small Java Virtual Machines (JVMs) that are available for 8-bit microprocessors (e.g., Ridgesoft, 2004). Here, the limited computing resources of a low-powered target platform are leveraged by using a desktop compiler that can creatively (and interactively) place the right resources on the target as they are needed.

At the top of the left column are small, interpreted languages that can be fully hosted on the robot platform. The Forth language is a good example (Hempel, 2004). The desktop computer is used only as a communications terminal, and code editing and compilation (if necessary) take place wholly on the robot target.

The 68HC11 Handy Board corresponds to the first column of this diagram. Aside from its set of hardware features, Interactive C distinguished the Handy Board from other 8-bit systems. IC made code writing more productive and debugging more effective.

Looking forward, the interesting part of the diagram is the middle column, but let's discuss the upper-right corner first. Here, the robot board is similar or equal in capability to the desktop workstation, and is running a full-fledged operating system with adequate RAM, a disk or flash-based file system, an installation of the gcc toolchain, etc. Perhaps the only thing it may be lacking is a keyboard and screen, so it may communicate via (e.g.) X-Windows with the desktop workstation. A research robot such as the ActivMedia's Pioneer robot (which runs the Linux operating system) takes this approach.

The "robot-is-a-Linux-computer" approach is obviously powerful, but we will advocate a more minimalist design. Rationally, we can make a cost argument—it will be cheaper to build a smaller computer that can't replace the desktop. But our motivation is more aesthetic; we are hoping to do a lot with a little, to create a platform that is still simple and yet extremely capable.

So, let's consider the middle column, which represents the "sweet spot" in the diagram. Here, we have a robot controller board that is much more powerful than an 8-bit micro, but would be stretched to run a full operating system such as Linux. There are two different software development possibilities here.

In the approach indicated in center of the diagram, we install a small operating system on the robot board, and use a desktop cross-compiler as a development tool. For example, the eCos operating system (eCos, 2004) provides a boot loader, a driver framework, memory allocation, thread management, file I/O, and a TCP/IP stack. Based on these resources, users can develop their own code in C or C++ using gcc on the desktop, and then compile, link, download, and run their programs.

The other possibility is represented in the top box of the middle column. The small OS is used as a foundation for running a significant interpreted environment, such as Lisp or Smalltalk, which hosts its own development environment. The OS provides needed services to these applications; the gcc toolchain is used to compile to build the kernel of the interpreted environment.

These two development scenarios are then our targets for the next generation Handy Board design. While not as general as a full modern operating system, a board that can support these two development modes would a vast improvement over the existing design, and would provide many opportunities for educators, students, and engineers.

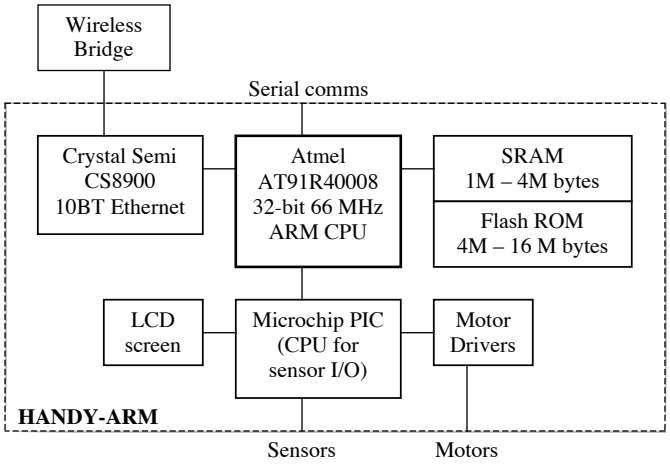

Figure 3: Block Diagram of Handy-Arm Board

Some Implementation Details

In this section we present our current implementation plan, illustrated in Figure 3. We welcome new ideas, criticism, and any other input.

With "Handy-ARM," we've already given away our choice of CPU—the 32-bit ARM processor. The ARM is a highly respected RISC processor family with a clean, orthogonal instruction set, implementations from multiple vendors, and a superb CPU-speed-to-power-consumption ratio.

The particular variant that we are evaluating (the 66 MHz Atmel AT91R40008) includes 256K bytes of internal static RAM, but for added flexibility, we plan to add between 1 to 4 Mbytes of external static RAM. This RAM, along with the Atmel's own, will be battery-backed. This feature will be especially important when using an interpreted development environment, which holds so much of its state in RAM. It will also be useful for performing data collection with the board.

With this battery-backed RAM, it would be possible to use RAM exclusively for user code, but an external ROM would still be required for a boot monitor. For maximum flexibility, we plan between 4 and 16 Mbytes of external flash ROM.

We plan TCP/IP communications capability by adding a low-cost Ethernet chip, the Crystal Semiconductor CS8900. This is an older chip that originally came into use on 16-bit ISA bus cards for PCs. It is now popular as an inexpensive and simple solution for adding Ethernet capability to embedded devices: Because it was designed for the simple 16-bit PC world, it's electrically easy to interface, and open-source drivers for it are widely available for many operating systems (including eCos).

Wireless Ethernet will be possible using a consumer-grade wireless Ethernet bridge device. (These presently retail for about $60.)

We would prefer a more integrated wireless solution, but the above plan is the most promising in terms of (1) driver development time, and (2) long-term availability of the components used in the design. Drivers are available for certain specific wireless cards, but none of these cards enjoys the ubiquity of the CS8900 device. We think the CS8900 is a good bet for being available in the years to come.

For the robot sensor and actuator I/O, we plan to use a separate low-cost CPU (the Microchip PIC). This will serve two purposes: (1) providing features that the Atmel does not have, such as analog-to-digital (A/D) conversion, and also (2) electrically isolating robot circuits from the main processor. This second aspect is especially important; if the design is done properly, voltage spikes and other disturbances from the robot will not cause hardware failures in the main CPU circuit. The I/O processor will also interface with low-speed devices like an LCD screen.

Closing Comments

Our controversial claim is that is more interesting to build a smaller system than a fuller one. We could defend this on cost grounds, but (even though this is a benefit) this is not the strongest argument.

The reason to build a simpler system is precisely that it is simpler. The result will be a design that is easier to implement and easier to understand, as well as having greater longevity and being less expensive.

Ease of understanding is a key quality of the existing Handy Board, and is valuable both for its pedagogical applications and practical engineering ones. When a design is open and documented, students can learn by understanding how circuits accomplish their work, and can design extensions and advanced projects based on this knowledge.

With the right design choices, a simpler design can be more useful than a complex one.

References

eCos operating system, http://sources.redhat.com/ecos/

Hempel, Ralph. pbForth programming environment, http://www.hempeldesigngroup.com/lego/pbForth/

Klassner, F. and Anderson, S., (2002). "LEGO MindStorms: Not Just for K–12 Anymore," *IEEE Robotics and Automation, Special Edition on Education-I*, June 2003, v 9, no 2.

Martin, F. (1994). *Circuits to Control: Learning Engineering by Designing LEGO Robots*, unpublished PhD dissertation, Massachusetts Institute of Technology, Program in Media Arts and Sciences, June 1994.

Martin, F. (2001). *Robotic Explorations: A Hands-On Introduction to Engineering*, Prentice-Hall.

Parallax, Inc (2004). Makers of the BASIC Stamp. http://www.parallax.com/.

Ridgesoft (2004). RoboJDE Java-based software development environment for the Handy Board. http://www.ridgesoft.com.

Sargent, R. (2004). *Interactive C Software*. http://www.kipr.org/ic/.