Analysis of a Danaher / Shouptronic 1242 Electronic Voting Machine

Undergraduate Independent Study
Final Report (Draft)
May 2008

Joseph Siefers
Lehigh University

Project Advisor: Professor Daniel Lopresti
Department of Computer Science and Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015 USA

Section 1: Obtaining the Object Code

Although the software that operates the Danaher 1242 is not publicly available ("closed-source"), an attacker with a small amount of funds, access to the machine, and a basic knowledge of computer architecture can obtain the object code[1] from the on-board PROM.

The Hardware

The first step in obtaining the object code is to partially dismantle the machine and gain access to the motherboard. Prior work by a fellow student exposed the motherboard, but a summary of that work follows. From the back of the machine, a service panel contains an election official control panel. After removing a few screws from the front of the machine, the panel can be removed in one piece. The mother board is directly adjacent to the control panel (Figure 2), and did not have any other protective barriers barring access.

Figure 1: Danaher Service Panel

Figure 2: Service Panel Exposing Motherboard

The Danaher 1242 utilizes the 27C256-20 EPROM (Figure 3), which contains 32KiB of object code. Most embedded systems have "socketed" chips in order to make updates and revisions to the hardware and software simple, and the 1242 is no exception. Revisions to the software of embedded systems are not as straightforward as with a general-purpose computer; the PROM that contains the program data must be removed and reprogrammed ("burned"). Unfortunately, this underlying

[1] Object code—the compiled version of the source code ready for execution.

characteristic of embedded computers creates a security risk in this particular application. In theory, an attacker could remove this chip and insert a new one of exactly the same type (assumedly containing additional malicious code with the original program code), without any apparent differences in the function of the machine. Some of the more recent machines place a tamper evident seal over the ROM to protect its integrity. However, the Danaher did not have any such safeguard—removal of the 27C256 required only a set of pliers. If the program code needs to be changed, a new PROM will have to be used, but this is not a serious problem because PROMS are available from electronic suppliers such as Digikey.

Figure 3: 27C256 EPROM

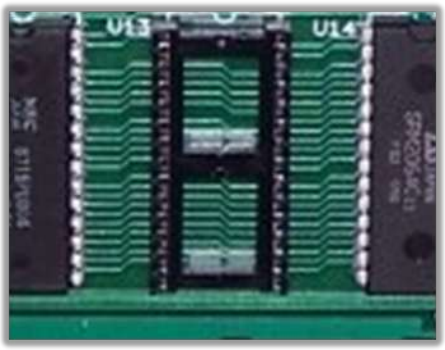


Figure 4: PROM Socket

The Interface

After removing the PROM from the motherboard of the machine, the next step is to develop an interface to read the data off of the PROM onto the host computer[2]. Surprisingly, one does not need to purchase an expensive PROM reader, which costs anywhere from $500-$1000, to read the data off of a PROM. Instead, he or she could develop C code for a properly configured microcontroller evaluation board (EVB). The semiconductor industry designs and produces EVBs to reduce development costs of microcontroller products, and in many cases, EVBs are sold below cost in order to stimulate sales of a particular microcontroller line. Therefore, a custom solution using an EVB typically costs much less than

[2] In the terminology of embedded computers, the host computer is the general purpose computer used to develop and compile the object code for an embedded computer. In this case, the host computer was used to analyze the existing object code.

a commercial PROM reader, and the attacker would only need to spend a small amount of time in development. The Dragon 12 (Figure X) uses the Motorola MC9S12DP256 microcontroller, a derivative of the Motorola HC12. The entire development kit, which includes an advanced real-time debugging interface, costs about $150 from www.evbplus.com.

The Dragon 12 has four banks of I/O pins which facilitate temporary connections between the processor and external devices. To read the program data, the address lines of the PROM were connected to the pins corresponding to ports A and K on the MC9S12DP256 (Figure X: green wires). The PROM data output lines were connected to Port B (Figure X: yellow wires). The serial port ("SCIO") was connected to the host computer that would receive the data.

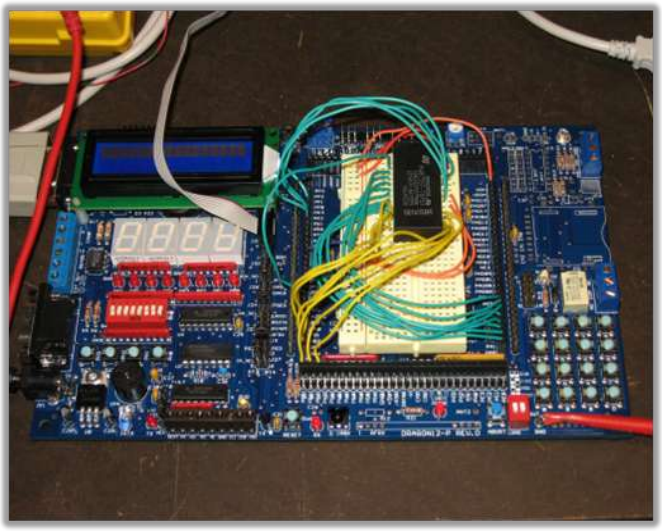


Figure XX: The Dragon 12 Interface

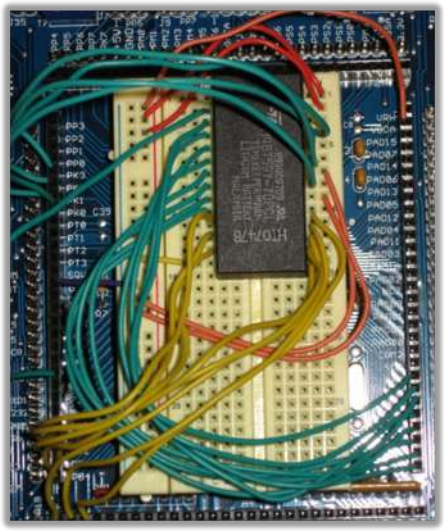


Figure XX: PROM fixture

Development of the C code to read the PROM was straightforward because of the wealth of knowledge provided by the MC9S12DP256 documentation (which is freely available from the Freescale website). Also, the real-time debugging interface provided with Freescale CodeWarrior made for easy testing. The code begins by mapping the MC9S12DP256 ports to their respective register locations and initializing the serial port registers with the proper control codes. Then, using nested for loops; it drives the address lines (Ports A and K) with the current address, and reads in the byte of data (Port B). Since

the MC9S12DP256 has less than 16KiB of RAM, the code immediately writes the current byte to the serial port. HyperTerminal received the bytes using the “Capture Text” feature, which conveniently stored the data in a binary file. The full C source is provided in the appendix.

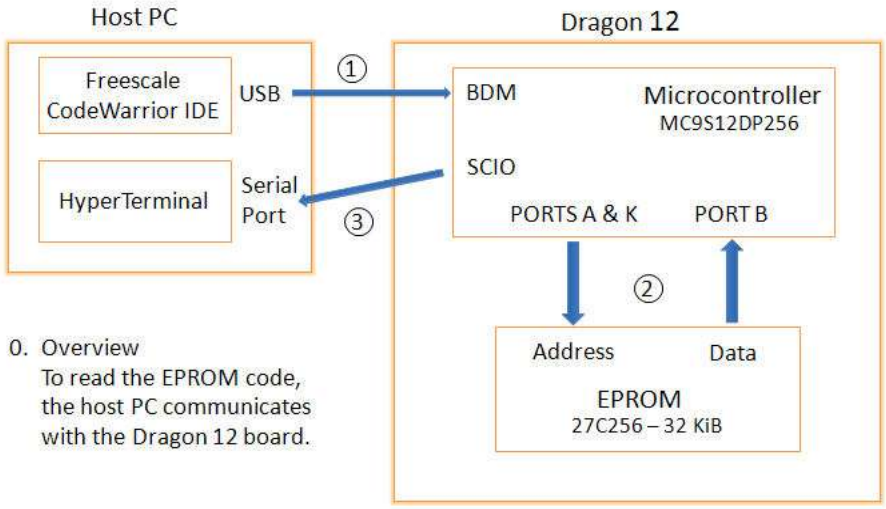


Figure XX: Dragon 12 Interface

Section 2: Interpreting the Object Code

Although the object code is not as easily to interpret as the source code, one can typically make sense out of it by through careful analysis in several steps. To start, one must understand the general disassembly process applied to any object code.

The Disassembly Process

The general disassembly process is given in Figure X describes the series of steps required to obtain a human-readable form of the object code. Note that the original source code in most cases is unrecoverable because a compiler eliminates most (if not all) of the abstractions that form the basis of high level languages before writing the actual machine instructions. The only choice, then, is to return to the interpretation of basic assembly. To make matters worse, the compiler typically also writes obscure sequences of instructions purely for the sake of code optimization or because it made writing

the compiler easier. Thankfully, a wide variety of software tools aid and guide the reverse engineer towards comprehension of vague sequences.

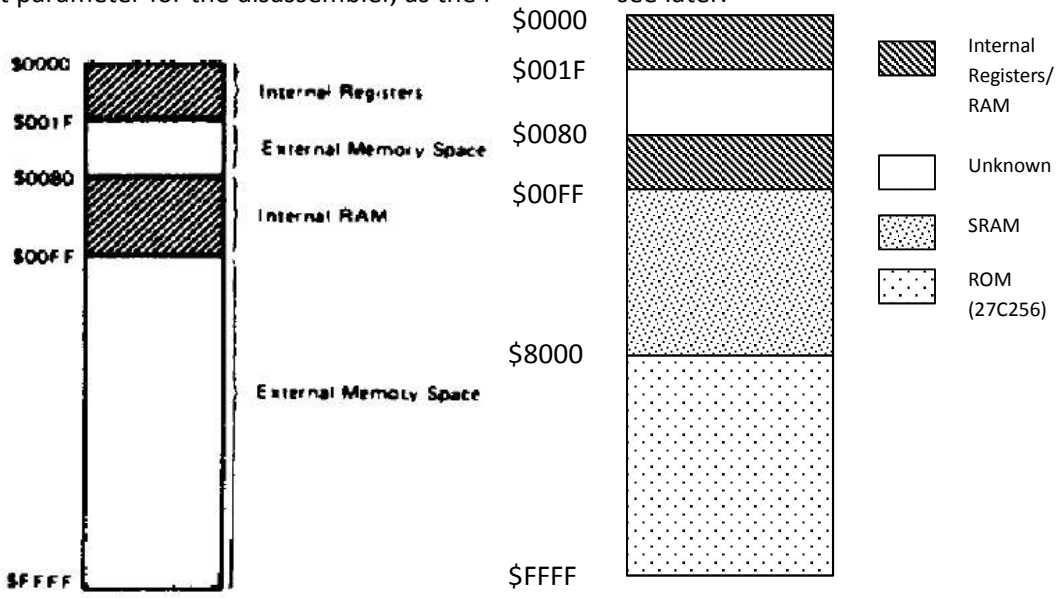
One such tool is known as a disassembler, and its function is to take the binary version of the object code and convert it into a human-readable form (assembly). It performs the exact opposite function as an assembler, but requires a significant amount of assistance from the user. The primary barrier it must repeatedly overcome is the inherent ambiguity of binary data—any particular binary sequence could either be a sequence machine instructions ("code") or information that the code operates on ("data"). In the analysis of the Danaher 1242 object code, this data is static because it resides in PROM that cannot be altered at runtime. The secondary barrier it must overcome is understanding the logical sequence of all the instructions that form the actual program. This concept is known as *tracing the execution sequence*, and requires knowledge about the way the processor interacts with the available memory in the embedded system.

Memory Map

Discovering the memory map yields information about the location of the object code relative to other devices in the system—RAM, UARTs, peripheral devices, etc—and is the first step towards completing an execution trace. The general memory map for the HD6303 is given in Figure XX, which shows its layout for "multiplexed" mode. Note that the Danaher 1242, or any embedded system for that matter, has its own specific memory map which is derived from the general memory map laid out by the microcontroller manufacturer. The major differences between the two maps result from how the large external memory space is divided among the external devices. The derived memory map for the Danaher 1242[3] is shown in Figure XX, and the schematic diagram for the actual wiring of the memories

[3] Note that at time of writing the complete memory map is not known because of difficulties in locating the destination of traces on the printed circuit board. Therefore, the locations shown are estimates based on the results of all experiments to date.

is given in the Appendix. The program memory (ROM) exists within $8000 and $FFFF, which is an important parameter for the disassembler, as the reader will see later.



"Multiplexed" and "non-multiplexed" mode define how the processor balances its most scare resource—pins—with the potentially large number of external devices it connects to. It is important to keep in mind that embedded systems typically address peripheral devices within the memory map as the actual external memory devices (such as RAM and ROM). Although this sometimes can become confusing, it simplifies the design (and programming) of the microcontroller.

The Danaher 1242 operates in "Multiplexed" mode, so that it can interact with as many peripherals as possible while still maintaining a large memory space. Memory is typically the largest consumer of pins, and is usually the target of any multiplexing operations. Effectively, the HD6303 sacrifices one pin[4] to make eight address pins both address and data pins. A latch[5] is then used to hold the address constant while the memory drives the data into the processor. While an additional eight pins may sound insignificant and not worth the hassle, most peripheral devices only require one or two pins, so this technique extends the capabilities of the microcontroller considerably. The collection of all

[4] The Address Strobe (AS) line.
[5] The SN74HC373N, which is located directly to the left of the processor.

the address and data lines that control all of the peripheral devices is known as the system bus. In Figure XX, which shows the pinout of the HD6303, the address and data lines are on the right side of the processor. The address lines on the left hand side are not used because of multiplex mode.

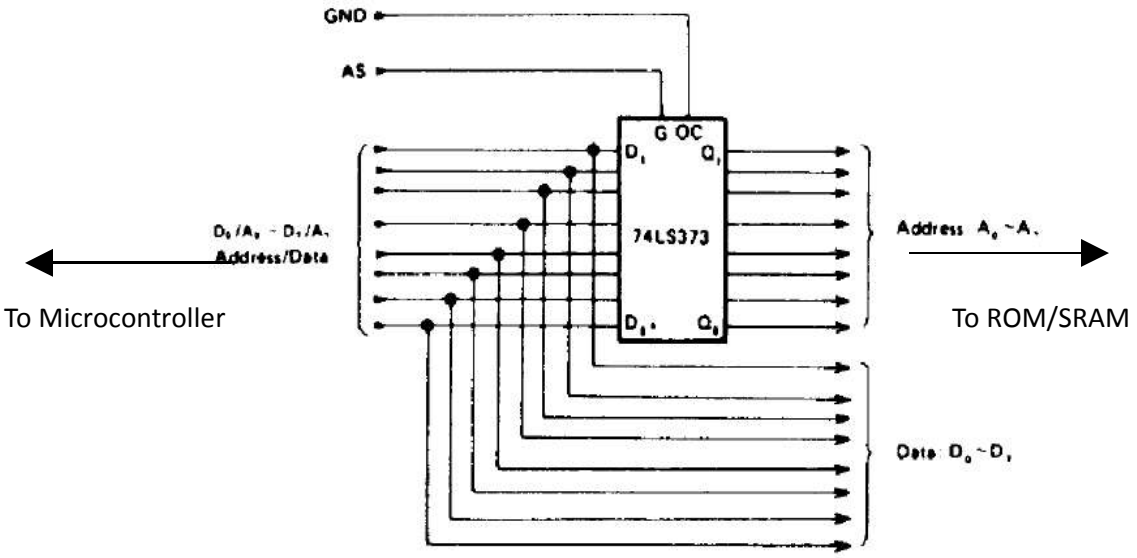


Figure XX: Latch Interaction with System Bus

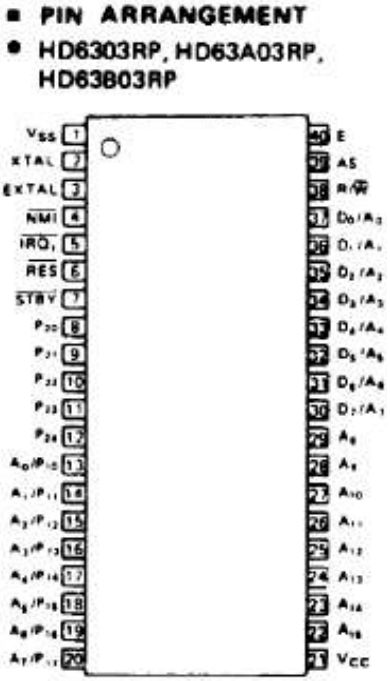


Figure XX: The HD6303 Pin Arrangement

Disassembly

Although the Danaher is equipped with an older microcontroller, the Hitachi HD6303RP, disassembly of the program code is still possible because the 6303 series shares a common architecture and instruction set with the Motorola 6803 series microcontroller. Experiments for this paper were conducted with DASMx 1.40, although those in the future should use IDA Pro (see Future Study). Identifying the entry points for the execution of the program code came directly from the datasheet for the microcontroller interrupt tables, and is the key for finding the execution stream.

Table 1 Interrupt Vectoring memory map

	Vector		Interrupt
	MSB	LSB	
Highest Priority	FFFE	FFFF	RES
	FFEE	FFEF	TRAP
	FFFC	FFFD	NMI
	FFFA	FFFB	Software Interrupt (SWI)
	FFF8	FFF9	IRQ, (or IS3)
	FFF6	FFF7	ICF (Timer Input Capture)
	FFF4	FFF5	OCF (Timer Output Compare)
	FFF2	FFF3	TOF (Timer Overflow)
Lowest Priority	FFF0	FFF1	SCI (RDRF + ORFE + TDRE)

Figure XX: Interrupt Vector Map

Section 3: Future Study

The first and most important/difficult task in order to completely disassemble the object code is to complete the memory map. At the time of writing, insufficient evidence was available to completely describe the circuit diagram (Appendix A.2). Understanding the complete memory map will provide the basis for completely fleshing out the execution stream. Progress on the disassembly is currently limited by an instruction which apparently jumps outside the PROM. Identifying the location of that address could complete the puzzle.

The second task is to accurately characterize the instruction set. Experiments using DASMx 1.4 in this paper were seemingly ineffective, and compatibility of the MC6803 instructions with the HD6303 was largely taken on faith according to the advice from developers of the disassembler programs. Purchasing IDA Pro (Appendix A.4) would be a worthwhile investment that would improve analysis significantly because of its interactive analysis functionality. The capabilities of IDA Pro allow a developer to characterize a *custom* instruction set for *any* processor—so compatibility with the HD6303 is *guaranteed*.

Once the code has been correctly disassembled and the execution stream is visible, the next task is to analyze the code for security issues. Understanding the interaction between the microcontroller and the button matrix would be useful for implementing an attack such as the secret knock attack[6].

Section 4: Conclusion

This paper discussed preliminary research towards the goal of understanding the object code of the Danaher 1242. The reverse engineering process is complex and time consuming, largely due to the fact that the documentation for the devices in the 1242 is brief and/or outdated. Also, manually tracing lines on a printed circuit board is tedious work that can become frustrating rather quickly. Nevertheless, future study on this machine would yield valuable insight into its subtleties. True embedded systems are largely immune to the standard attacks of general purpose computers because they lack many of the prerequisite capabilities (such as internet functionality). However, modern voting machines are beginning to take on the characteristics of general purpose computers (they run windows or other standard operating systems). However, it is extremely important to understand the history of these machines in order to be able to improve future designs.

[6] The secret knock attack in theory is completely undetectable—the machine starts and behaves normally until the attacker presses button sequence on the front panel of the machine, in which case the machine begins to behave according to the malicious code inserted onto the compromised PROM.

Appendix

A.1 Acknowledgements

Professor Decker, Professor Lopresti.

A.2 Dragon 12 C Code (Attachment)

A.3 Circuit Diagrams

A.4 IDA Pro Information (Attachment)