

## PC Interfacing by Example

John Fulcher

*School of Information Technology and Computer Science, University of Wollongong, Northfields Ave, Wollongong NSW 2522, Australia*

*john@uow.edu.au*

**Keywords:** control technology, experimental, problem solving, programming, user interface

**Abstract:** A laboratory environment for the teaching of microcomputer interfacing and real time computing is described. An Integrated Development Environment (IDE) running on the host PC facilitates the writing of software for a target 68HC12 microcontroller, which connects to host via the latter's serial port. Students are encouraged to learn by doing, including learning from their mistakes – hence necessitating good debugging facilities within the IDE. This approach is in keeping with that of situated learning (Lave and Wenger 1991). To put it another way, knowledge as such is not *taught*, but rather imparted through a process of apprenticeship. In this context, the role of the educator is primarily to provide a nurturing environment in which students are able to get their hands dirty and attempt to interface to real-world peripheral devices. Lecturing staff are used as consultants or last resort trouble-shooters, with students encouraged to perform the bulk of their own software development and debugging. Experience has shown that students respond admirably to such an approach, commonly experiencing the subject as “it’s a lot of work, but very enjoyable – I learnt a lot”.

### 1. INTRODUCTION

We have a tradition at the University of Wollongong of designing our undergraduate subjects to be very much laboratory-based. This is particularly the case with the Microcomputer Interfacing and Real-Time Computing subject which students can elect to undertake during their third

and final year of their Bachelor of Computer Science study. We believe students learn by doing, *especially* by getting their hands dirty and learning from their own mistakes (Fulcher 1990). Furthermore, they best learn underlying principles by having to apply these to real-world problems (Fulcher 1991).

Another emphasis – in both subjects – is the exposure of students to what lies under the hood of the computer. In other words, they gain an appreciation of what is contained within this black box called a computer.

Introductory subjects in computer architecture, computer systems and/or assembly language programming tend to use simulators (e.g. Gray 1987, Patterson and Hennessy 1994). By contrast, we have recently shifted the focus of our introductory computer systems subject to that of PCs in general, and the Pentium processor in particular (Duntemann 2000) – one reason for this, by the way, is that this is the first time the author has encountered PC assembly language programming covered not only from a DOS perspective, but also for Linux.

The shareware assembler NASM forms the common link between DOS and Linux in the PC environment (an entire NASM-IDE in the case of the former). The inbuilt (16-bit) debug is used in the DOS environment, whereas the (32-bit) gcc debug tool – gdb – is used under Linux (which causes a slight trauma in students migrating from the Intel-style syntax of debug to the AT&T syntax used in the gcc tool suite!). Furthermore, reliance on BIOS software routines under DOS shifts to standard C library function calls under Linux.

The PC emphasis of the earlier Introduction to Computer Systems subject is expanded upon in the third-year Microcomputer Interfacing and Real-Time Computing subject. In the latter, students are given ample opportunity to control real-world devices (not just *simulations* thereof). Feedback from students is invariably positive – they typically cite this subject as one of the best they undertake during their entire 3 years Bachelor of Computer Science degree studies.

Enrolment numbers are typically between 30 and 40, compared with 120 to 150 in the first-year Introduction to Computer Systems subject.

## 2. LABORATORY SETUP

Previous incarnations of the Microcomputer Interfacing Laboratory were based around the 8-bit Motorola MC6800 (via Heathkit ET3400 trainers – Fulcher 1989a), and the 16/32-bit MC68000 (via firstly Motorola's Educational Computer Board, thence the Applix 1616 – Fulcher 1989b,

Fulcher 1990, Fulcher 1991). In each case, the basic microcomputer trainer interfaced to individual experiment pods via in-house expansion circuitry.

In choosing individual (plug-in) experiment pods, we were conscious of making the laboratory assignment work both interesting and accessible to the students. Typical experiments include:

1. LCD/Dot Matrix displays
2. timer/music
3. serial communications
4. bar code reader
5. x-y drill positioner
6. slot car controller
7. model train scheduler, and
8. turtle robot controller.

In each case, heavy emphasis is placed on students writing their own (rudimentary) software drivers for the hardware controllers provided. To quote from the preface of an earlier accompanying textbook (Fulcher 1989a): “Interfacing peripheral devices to a microcomputer involves three important factors. *Firstly*, we need to know the basic operating principles of the peripheral device we’re attempting to interface to the computer. *Secondly*, we need to know the characteristics of the particular peripheral support chip we’re using for this task. *Thirdly*, we need to write the control software.”

The immediate feedback provided to students by having LEDs flash on/(off), interrupts being generated, and the actual movement of real world devices greatly assists students in their learning experience.

A few years ago we decided to migrate our existing (MC68000-based) experiment pods to a PC platform. One motivation behind this decision was to impart skills to students which they would be able to take *beyond* the classroom laboratory setting and into the real world following completion of their formal studies.

In order to ease the transition to a PC environment, we developed an in-house ISA expansion card, which allowed direct connection to our pre-existing (MC68000) range of experiment pods. The following year saw these older experiment pods replaced by a single development platform based around a Motorola MC68HC12 microcomputer.

### 3. LABORATORY ASSIGNMENTS

The first PC-based laboratory assignments were undertaken on the PC itself, using the Open Source DJGPP C/C++ Compiler (in fact a complete Integrated Development Environment – <http://www.delorie.com/djgpp>).

Open Source software was chosen rather than commercial, in order to keep costs low (e.g. Microsoft Visual C++ – Buchanan 1996).

Extensive use is made of interrupts – both hardware (from external peripherals) and software. The latter focuses on BIOS library function calls, with which students become familiar during the (first-year) Introduction to Computer Systems subject. It is only in the later-year subject however that the separation of functions contained within the *highly integrated* PC support chips becomes clear. Indeed, a complete system can be constructed using just three (highly integrated) peripheral support chips:

1. i82439 System Controller (DRAM, SRAM/cache, ECC),
2. i82371 PCI/ISA Xcelerator (IDE, PCI-ISA bridge, Priority Interrupt Controller, timers, plug-and-play, mice, USB root hub), and
3. i82091 Advanced Interface to Peripherals (2 serial, 1 parallel, FDD).

In the later-year subject, interfacing is approached from a *functional* perspective, but again with a focus on interrupt-driven IO and buffering.

The mode of operation of the laboratory equipment was subsequently changed from code development on the target machine (i.e. the PC itself), to a PC host/68HC12 target configuration. A locally developed 68HC12 Monitor-in-RAM provides students with rudimentary functionality, although not as comprehensive as that contained within the PC BIOS:

- |                       |   |   |
|-----------------------|---|---|
| D<from, nbytes>       | - | <u>D</u> isplay memory                              |
| F<from, nbytes, data> | - | <u>F</u> ill memory                                 |
| T<from, nbytes>       | - | <u>T</u> est memory                                 |
| G[addr]               | - | <u>G</u> o from address (i.e. start executing from) |
| L                     | - | <u>L</u> oad record (i.e. *.S19 executable file)    |
| H                     | - | <u>H</u> elp (i.e. list of available commands)      |

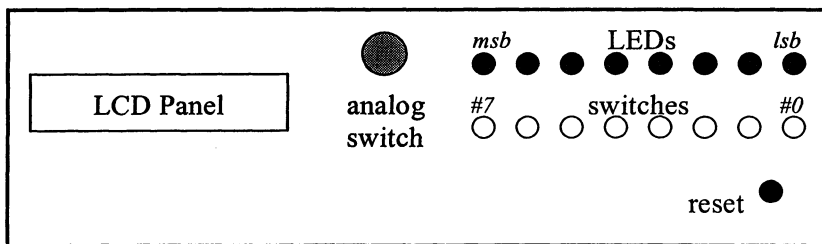


Figure 1. MC68HC12 Target Microcontroller Front Panel

The 68HC12-based microcontroller is shown in Figure 1. It cost around \$600 per unit to produce, and incorporates the following peripherals (with associated 68HC12 ports – see Figure 2):

1. 8 switches    Data Register=Port-D 0x05    Data Direction Reg-D=0x07

2. 8 LEDs Port-J 0x28 DDRJ 0x29  
 3. (2-line\*8-digit) LCD Panel Data: Port-H 0x24 DDRH 0x25  
 Control: Port-T 0xAE DDRT

0xAF

4. 5 timer channels (3-7) Port-T 0xAE DDRT 0xAF  
 5. Serial Interfaces (2 asynchronous SC0(1) + 1 synchronous SP0)  
 SC0(1): 0xC0-C7(0xC8-CF) SP0: 0xD0-D5  
 6. 8-channel (8-bit) Analog-to-Digital Converter Port-AD 0x70-7E  
 7. USB interface (*under development*)

The HC12 interfaces to the PC via the latter's RS232 serial port (an upgrade to USB is currently underway).

The 68HC12 microcontroller cross development platform takes the form of the ImageCraft C-compiler – ICC12. As with the DJGPP C/C++ Compiler, the Integrated Development Environment adopts the usual Borland style (i.e. integrated editor, compiler, linker and debugger, all accessible from the same Graphical User Interface).

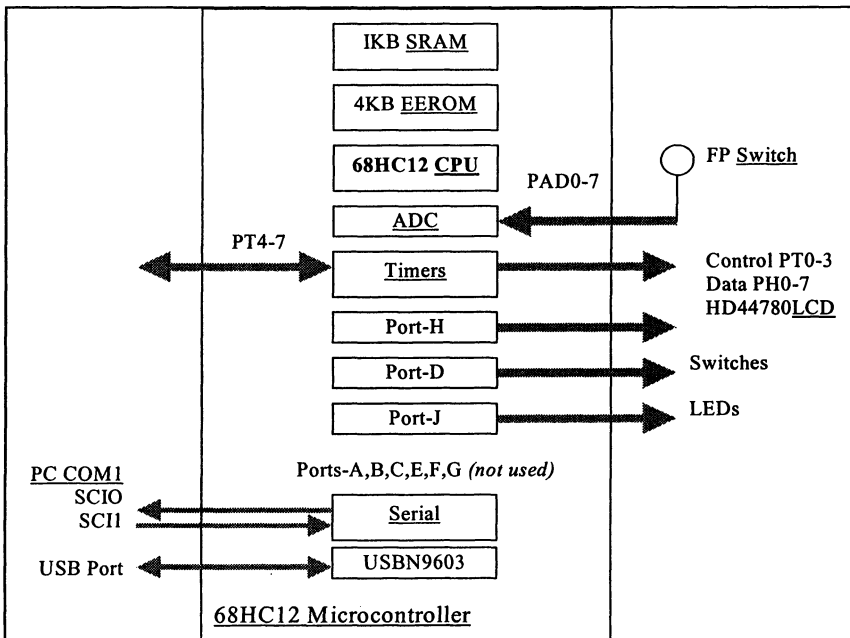


Figure 2. Motorola 68HC12 Microcontroller

Typical assignment tasks include the following:

1. Simple IO (switch input; LCD output):
  - activation of least significant (software debounced) switch(#0) → print name on LCD

Networking the Learner

Computers in Education

Watson, D.M.; Andersen, J. (Eds.)

2002, XXXIV, 991 p., Hardcover

ISBN: 978-1-4020-7132-4