Microcontroller Lab Hardware

Günther Gridling

February 27, 2006

Abstract

This report\textsuperscript{1} describes the hardware designed for an undergraduate course ‘Microcontroller’, in which students are familiarized with microcontroller and low-level programming.

1 Introduction

Our department offers an undergraduate course ‘Microcontroller VL’, which aims to give students an introduction into software development for microcontrollers. Naturally, this involves much low-level programming, which most students are not familiar with, so they need a lot of time to practise. Since our lab, being shared with other courses, can only accommodate at most 10 students at a time, and there are about 150 of them, it was decided to offer so-called ‘lab kits’, which contain the basic equipment required for the lab work. Students can take the kits home, which gives them the opportunity to work on their tasks according to their own schedule.

2 Requirements

In 2003, when this course was first set up, an extensive search for suitable off-the-shelf hardware was conducted. Apart from supporting all the course examples, the hardware needed to be low-cost, since the budget is always limited. The following requirements were identified:

- Although there are a lot of quite advanced simulators available, the idea of a course based completely on the use of a simulator was dismissed early on, mostly for two reasons. One is that beginners tend to misuse the comfortable debugging environment of a simulator. Instead of using it as the ideal tool to help them understand the inner workings of the controller by allowing them to observe what is going on at any time, they often end up following a pure trial-and-error approach: Compile and run; watch registers, variables, memory, stack, just about everything you can fit on the screen;

\textsuperscript{1}This work was supported by the SCDL “Seamless Campus: Distance Labs” project, which received support from the Austrian “FIT-IT Embedded Systems” initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Research Promotion Agency (FFG) under grant 808210. See http://www.ecs.tuwien.ac.at/Projects/SCDL/ for further information.
if something seems wrong, modify ...well, anything in the immediate vicinity; repeat until the program seems to work and see if you get away with it.

In contrast, having nothing but a few LEDs as a monitoring device forces students to analyze the program’s logic more closely. With the hardware being basically a black box, they often quite naturally realize that a foggy idea of the controller’s behaviour is not enough, but that one needs to understand what exactly is going on in there.

A second reason not to use a simulator is that computer science students, even in embedded systems branches of the curriculum, are often completely unfamiliar with hardware, save for the box under their desks that is referred to as a PC. Of course, they’re not EE students, but they should still have a passing familiarity with the common electronic components and should at least be able to recognize them upon sight. They should be aware of common phenomena like switch bouncing and floating inputs, they should know that interrupts have a latency, that a 12 bit wide ADC data register does not necessarily imply that the result is 12 bit accurate, or that the output ports of a microcontroller cannot change the state of the lines in zero time. A lot of the peculiarities of hardware designs are best observed on actual hardware instead of just a simulator. It seems that many students need the experience of long and frustrating debugging sessions in order to really appreciate all the seemingly arcane facts in the datasheet, which they initially ignored.

- To avoid being limited to one microcontroller architecture, the hardware needed to be modular, consisting of at least two boards: One containing the microcontroller itself, and one with the basic I/O-hardware. Additional boards with more complex I/O (at least an RS-232 adapter and a motor driver) would also be needed.

- Atmel’s AVR microcontroller family should be supported, since this architecture with its clear and well documented design was deemed best suited for course work. Also, AVR controllers and demo boards are readily available at most vendors. This was important so that students who might get interested to do home projects could easily buy similar hardware.

- Connections between the boards had to be flexible in that it should be possible to connect any microcontroller port pin to any I/O element, rather than having fixed connections. Students should have to decide exactly how the various components have to be connected to the controller. To some extent, this requirement was triggered by the traumatic experience of students asking whether they had to connect both ‘ends’ of a LED in order for it to light up. With fixed connections, they learn that a LED can be switched on and off by toggling a particular bit in some register, but they might go on thinking it was probably just one ‘end’ of the LED – after all, it was only one bit.

- A certain robustness was required, i.e., it should not be easily possible to damage LEDs or the microcontroller by making wrong connections, which happens quite often.

As for I/O elements, the following was identified as the minimum requirement to support the course goals:

- A multiplexed numeric LED display with at least four digits. Students should be familiar with the principle of multiplexing.
A matrix keypad of 4x4 buttons. Again, scanning a matrix keypad is a task often encountered and is quite instructive. In combination with the numeric display, a keypad makes many interesting programming exercises possible.

- Two separate buttons, one of them hardware-debounced.
- Eight LEDs.
- Two switches.
- One potentiometer.
- One photo transistor.

Unfortunately, back in 2003, no off-the-shelf hardware was found which fulfilled all the requirements and was still affordable. Therefore, it was decided to design dedicated hardware for this course.

3 The Microcontroller Board

As required, the microcontroller board contains an Atmel AVR microcontroller, in particular the ATmega16 [2]. This controller has all the basic features:

- Four bidirectional I/O ports with optional pull-ups.
- 16 KB of in-System programmable FLASH, 1 KB SRAM, 0.5 KB EEPROM.
- 8 and 16 bit timers/counters.
- External interrupts.
- ADC, analog comparator.
- USART, SPI, TWI.
- PWM.
- A watchdog.
- Various sleep modes.

All the I/O pins of the controller are accessible through sockets, so that they can be connected to I/O boards.

In addition to the controller, the board contains a voltage regulator as well as series resistors between the port pins and the corresponding sockets to protect against accidental shorts.

Originally, the microcontroller board was designed to use a standard Atmel ISP-Header, requiring the use of an ISP-Adapter on the serial port of the PC. More students than expected, however, use a laptop PC exclusively, which these days usually do not feature a serial port anymore. Common USB ↔ RS-232 adapters, however, do not work for ISP programming. Therefore, it was decided to adapt the board such that programming via USB was
possible. In addition to that, serial interfacing would have to be done via a built-in USB ↔ RS-232 adapter.

To be as flexible as possible with regard to the USB interface, it was decided to use a microcontroller with built-in USB interface rather than some dedicated RS-232 ↔ USB converter IC like the FTDI FT232 series. The requirements were as follows:

1. Full-speed USB with bulk transfer mode.
2. Preferably, the controller should be in-system programmable.
3. The package should be one that can be easily hand-soldered (ideally SOIC).

In addition, hardware SPI and USART would be desirable, but not mandatory, since both can be easily bit-banged.

A number of options were considered, in particular Microchip PIC18F2455, Atmel AT89C513xA, and Cypress CY7C64013A. Ultimately, the AT89C5131A [1] was chosen, since it fulfilled all the requirements and was readily available at an acceptable price.

In the schematic, the USB controller appears in the upper left corner. Since the AT89C5131 has built-in USB transceivers, not many external components are required:

1. The USB-B connector and the usual series and pull-up resistors, with an optional inductor to filter the +5 V line, in case the board is powered via USB.
2. One SMD jumper with capacitor and series resistor for the reset line.
3. A 16 MHz crystal with capacitors.
4. Two capacitors and one resistor as the external components for the controller’s PLL, which produces the 48 MHz clock needed for USB.
5. Four series resistors on the lines that are used to program the ATmega16 via SPI (three SPI signals plus the ATmega16 reset line).
6. Four LEDs on the controller’s current-source outputs, used by the firmware to signal programming and RS-232 activity.
7. Two series resistors protect the USART pins from accidental shorts due to programming errors.
8. Six I/O pins are accessible via optional sockets.

The USB controller’s firmware has three main functions:

1. In-system programming and verification of the AT89C5131 itself.
2. In-system programming and verification of the ATmega16.
3. Providing a RS-232 ↔ USB converter, so that the ATmega16 can communicate with the host PC using its USART.
To the right of the USB controller are the headers for in-system programming and JTAG, which follow Atmel’s pinout specifications. The ISP is an optional backup, as programming is usually done through USB. Note that the resistors on the socket side of each of the corresponding pins only protect against the external I/O which might be connected to those pins. With the ISP header, this is not much of a problem, as the programming adapter pulls reset low, disabling any user program which might be driving the ISP pins. If JTAG hardware is connected, however, there is a possibility of contention in case the user forgets to enable the JTAG fuse and the currently active user program happens to drives the JTAG pins. The JTAG adapter would have to provide appropriate protection.

There are two voltage regulators on the controller board. 5 volt are required for the ATmega16, while the USB controller needs 3.3 volt. Both are normally powered with 7.5 volt from an external wall wart. However, there are other power supply options, selected with three jumpers VREG, SEL33, and BPWR:

<table>
<thead>
<tr>
<th>VREG</th>
<th>SEL33</th>
<th>BPWR</th>
<th>Power Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>closed</td>
<td>right</td>
<td>open</td>
<td>7.5 volt wall wart</td>
</tr>
<tr>
<td>open</td>
<td>left</td>
<td>open</td>
<td>external 5 volt connected to any VCC socket</td>
</tr>
<tr>
<td>open</td>
<td>right</td>
<td>closed</td>
<td>5 volt from USB bus</td>
</tr>
</tbody>
</table>

Note that according to the datasheet [12], the 3.3 V regulator expects at least 5.3 V at its input, so both the second and third configuration are actually beyond specification. However, since the AT89C5131 does not draw much current, it should still work with a clean 5 V supply.

Both VCC and GND are available at several sockets, as is the unregulated 7.5 voltage from the external power supply on the six PWR sockets.

The lower half of the schematic shows the ATmega16 itself: External components are just the reset-button with pull-up and capacitor, 16 MHz crystal with capacitors, and the usual decoupling capacitors. Furthermore, each port pin is connected to a socket for external access, with a series resistor in between. These resistor arrays are socketed, in case they ever need to be adapted to a special application. This is, of course, especially important for port A, where each pin can be used as an ADC input. Also, the ADC’s reference voltage input pin AREF is externally accessible. Note that care must be taken when connecting an external voltage reference: As described in the datasheet, if the controller is configured for internal voltage reference or AVCC as reference, those are physically connected to the AREF pin (so that external filtering is possible). If at the same time an external reference is connected, the two voltages will be shorted out. For details, see the datasheet [2], page 209f.

A note regarding the layout of the boards: The clock frequencies on these boards is relatively low, so the layout is not particularly critical. It would most likely have been sufficient to let the autorouter do the 70% of traces it can manage and just route the rest manually. However, since these boards are to be used by students in a lab, it was desirable for the PCB layout to be decent enough in so far as the usual recommendations (proper placement, short tracks, few vias, as solid a ground plane as possible) should be followed. Therefore, all the boards were manually routed.

3.1 The I/O Board

The I/O board meets or exceeds the requirements stated above. It contains:
- A six digit multiplexed numeric LED display with darlington drivers.
- A matrix keypad of 4x4 buttons.
- Eight buttons, with BTN7 and BTN8 connecting to ground and BTN7 being debounced by a capacitor. BTN1-3 are connected to the same socket at one side, so the user can decide whether they switch to GND or VCC by making the appropriate connection. Same for BTN4-6.
- Eight LEDs.
- Eight switches, following the same connection scheme as the buttons, except that no switch is debounced.
- Two potentiometers.
- One photo transistor.

The schematic is rather straightforward. All sockets except those at the ULN2003A [13] darlington array inputs are protected by series resistors. This makes it possible for students to check a LED by simply connecting it to VCC/GND — or rather, it prevents damage to the LEDs when students do so.

The green power LED shows that the board is powered, and the red polarity LED alerts the student in case of reverse polarity. All the LEDs and seven-segment displays are low-power (requiring only about 2 mA), so the I/O board can easily be powered by the regulator on the ATmega16 board.

Again, the series resistors are socketed, so that the values can be adapted.

We usually give interested students the opportunity to buy the parts and solder a board themselves under our supervision. Each year, about 20-30% of the students take up that offer. For that reason, the I/O board (as well as the motor driver boards described below) uses through-hole components exclusively, since SMD components are a little demanding for someone who might have never before used a soldering iron.

### 3.2 The MC3479 Motor Driver Board

The MC3479 board can be used to drive a bipolar stepper motor. It uses OnSemi’s MC3479 [8] motor controller, which provides the students with a simple way to control a stepper motor.

All control signals are available at the IN connector. The power outputs are available at the OUT connector, and there is a standard connector for bipolar motors as well. Optional resistors can be included via jumpers, in case the motor exceeds the controller’s power rating.

### 3.3 The L293D Motor Driver Board

On the above board, the motor controller takes care of energizing the motor coils in the appropriate sequence. Students basically only need to supply step and dir signals, and they can quickly get a motor running. However, they should have at least once directly controlled
Figure 2: Basic I/O Board Schematic
Figure 3: MC3479 Motor Driver Board Schematic
Figure 4: L293D Motor Driver Board Schematic
a stepper motor. For that, the L293D [4] motor driver board is used. It contains a four-channel push-pull driver, which can be used as two independent H-bridges to drive stepper or DC motors.

The board has its own voltage regulator for the logic supply so that it can be powered with unregulated 7.5 V, as the L293D requires at least 4.5 V, but some of our controller boards only supply 3.3 V. HCT inverters with schmitt-trigger inputs are used as level translators.

3.4 The Interface Board

On the ATmega16 controller board, the use of USB is transparent to the students. It serves as the programming interface, and the ATmega16’s USART I/O is translated to USB for the host PC. However, during the MCVL course, the students should have the chance to actually implement USB communication. Furthermore, they should become acquainted with some other common communication interfaces, most notably CAN, RS-232, and RS-485.

The interface board provides all four of the communication interfaces mentioned above. For USB, FTDI’s FT245BM [3] USB FIFO is used, which provides an 8-bit bus interface to the microcontroller. The optional EEPROM can be used to implement a device ID and to return a USB 2.0 Full speed device descriptor instead of the default USB 1.1. The Maxim MAX232 [7] level translator can be used to connect the microcontrollers USART to a PC’s serial port (or even an old serial modem or terminal). Two TI SN75176B [11] differential bus transceivers are available for EIA/TIA-422-B or RS-485 Finally, the board offers a SJA1000 [10] stand-alone CAN controller together with two PCA82C250 [9] CAN transceivers. Like the FT245BM, the SJA1000 can be connected to the microcontroller via an 8-bit bus interface. One male and one female DB9 connector are available for use with RS-232/422/485 or CAN.

3.5 Power Supply Board

The controller board has its own power input with regulator, which can be connected to other boards. For motors, however, higher voltages than those 5 V might be required. Also, it makes sense to power motors and logic from different supplies. For that, the power supply board can be used.

There are three regulators with separate inputs for unregulated voltage on the board. Two of them are fixed voltage regulators of the 78XX series [5], which offer voltages between 5 and 24 V, output current rating of up to 2 A with heat sink, internal current limiting, and thermal protection. Even the most inexperienced student should be hard pressed to accidentally destroy one of these.

The third regulator is a LM317 [6], which is adjustable between 1.2 and 37 V and has an output rating of 1.5 A. Like the above, this regulator sports internal current limiting and thermal protection.

3.6 RS-232 Board

The current microcontroller board uses an USB controller to translate between the USART and USB. If, however, a direct RS-232 connection should be required, the RS-232 board can be used.
Figure 6: Power Board Schematic
Figure 7: RS-232 Schematic
Apart from the MAX232 [7] level translator, it offers two DB9 connectors, which are not hardwired to the MAX232, but can be connected via sockets to offer the flexibility to connect, e.g., signals like RTS and CTS in addition to TX and RX.

3.7 Serial Programming Adapter Board

The Serial Programming Adapter Board allows programming of the ATmega16 via a PC’s serial port, in case no USB port is available.

The ATmega’s SPI-pins used for programming are connected to the PC’s serial port via tri-state buffers. During programming, the reset line is pulled low, thereby enabling the buffers. This allows the programming software to drive the SPI lines. Normally, reset is high, which puts the buffers in high-impedance state. In this case, the SPI lines are not affected by the programmer.

The cope with possible noise, the signals coming from the PC are fed through schmitt-triggers. At each input, an optional capacitor can be added after the series resistor to provide an RC filter.
Figure 8: Serial Programming Adapter Board Schematic
References

[2] ATmega16(L) Datasheet, Rev. 2466L-06/05.