Microcontrollers

Andrew Sampson
2012-12-13

Introduction

This presentation is a brief overview of microcontroller software development, with a focus on the Atmel line of MCUs. I'll cover:

- how development for these devices differs from traditional desktop PC development
- the rules for our programming contest

I'll wrap up by handing out some development kits.

Microcontrollers

Microcontrollers (MCUs) are very low-end, inexpensive CPUs.

- They're used in all kinds of electronics products, and they're usually used in unglamorous, low-visibility roles.
- It's big business: wikipedia says that "About 55% of all CPUs sold in the world are 8-bit microcontrollers and microprocessors."

Microcontrollers

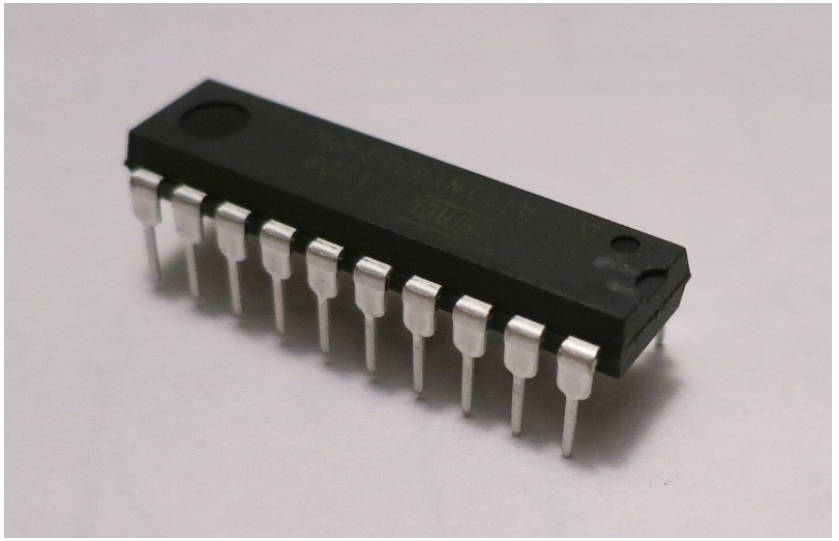
- Trends in MCUs have focused on consolidating more and more functionality in a single chip.
- While MCUs can't really compare with a desktop in performance, they do incorporate everything needed to run a simple program in a single chip:
  - CPU
  - RAM
  - storage
  - serial UART
  - USB and CAN interfaces
  - timers

Microcontrollers

Given the intended role of MCUs, and the fact that they don't depend on external chips, most of the pins on MCUs are "general-purpose". Here, this means that a pin's voltage state (high or low) can be *controlled directly* by the software running on the MCU. This is very different from how, say, a desktop PC CPU operates.

What's it like to write software for an MCU?

We're using an Atmel ATtiny2313
It has 2 kilobytes of (flash) program memory
It has 128 bytes of RAM
It has 128 bytes of EEPROM



What's it like to write software for an MCU?

Despite these severe limitations, programming the ATtiny2313 isn't really uncomfortable. You can use C. The biggest differences between programming this MCU and a desktop system are:

- There's no OS: no file system, no memory protection, etc
- There is a libc, but it's very basic
- malloc() doesn't really make sense here (128 bytes, remember)
- The stack is very short
- There's no FPU
- The ALU has no multiply or divide instructions
- The native integer length is 8 bits

Word length

The Atmel AVRs are 8-bit systems. For efficiency, you should stick with 8-bit data types as much as possible. However, the avr-gcc compiler does a good job of hiding the ugly if you need to use longer data types.

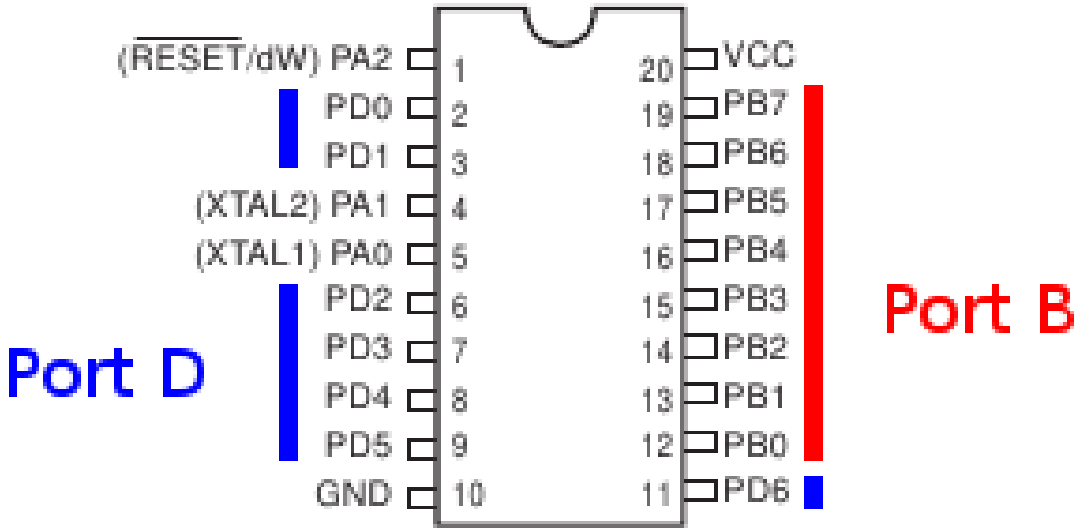
Learn to love the stdint.h data types, i.e. uint8_t, int16_t, etc.

Memory

- The AVRs follow the Harvard architecture, with separate memory spaces for instructions (flash) and data (RAM).
- The instruction memory is much larger than the data memory for these devices.
- It is preferable that large constants (strings, arrays) remain in flash, and not be copied to the stack.
- There are functions and macros – “PROGMEM” – for forcing a constant to remain in flash, and for accessing such a constant. See the links slide for details.

Pins

For Atmel MCUs, the pins are logically (and sometimes physically) organized into groups of 8 (called "ports").



Pins

Ports appear in software as special memory locations. Changing the high/low states of the pins is as simple as assigning a value to the memory location. Reading the state of the pins is equally simple.

Setting pin 5 on port D high:

```
PORTD |= (1 << 5);
```

Setting all pins on port D high:

```
PORTD = 0b11111111;
```

Setting pin 5 on port D low:

```
PORTD &= ~(1 << 5);
```

Reading all pins on port B:

```
uint8_t blah = PINB;
```

Configuring pins

Pins can be either inputs or outputs, but not both at the same time. You can set the input/output state of a pin by (you guessed it) writing to a memory location.

Set pin 5 of port D to be an output:

```
DDRD |= (1 << 5);
```

Set pin 5 of port D to be an input:

```
DDRD &= ~(1 << 5);
```

Set all pins of port B to be outputs:

```
DDRB = 0b11111111;
```

Configuring pins

Pins are configured as inputs by default. It's good practice to configure your pins to known input/output states at the beginning of your program. Also, because the pin I/O state is global, you need to manage it carefully (esp. when it comes to functions).

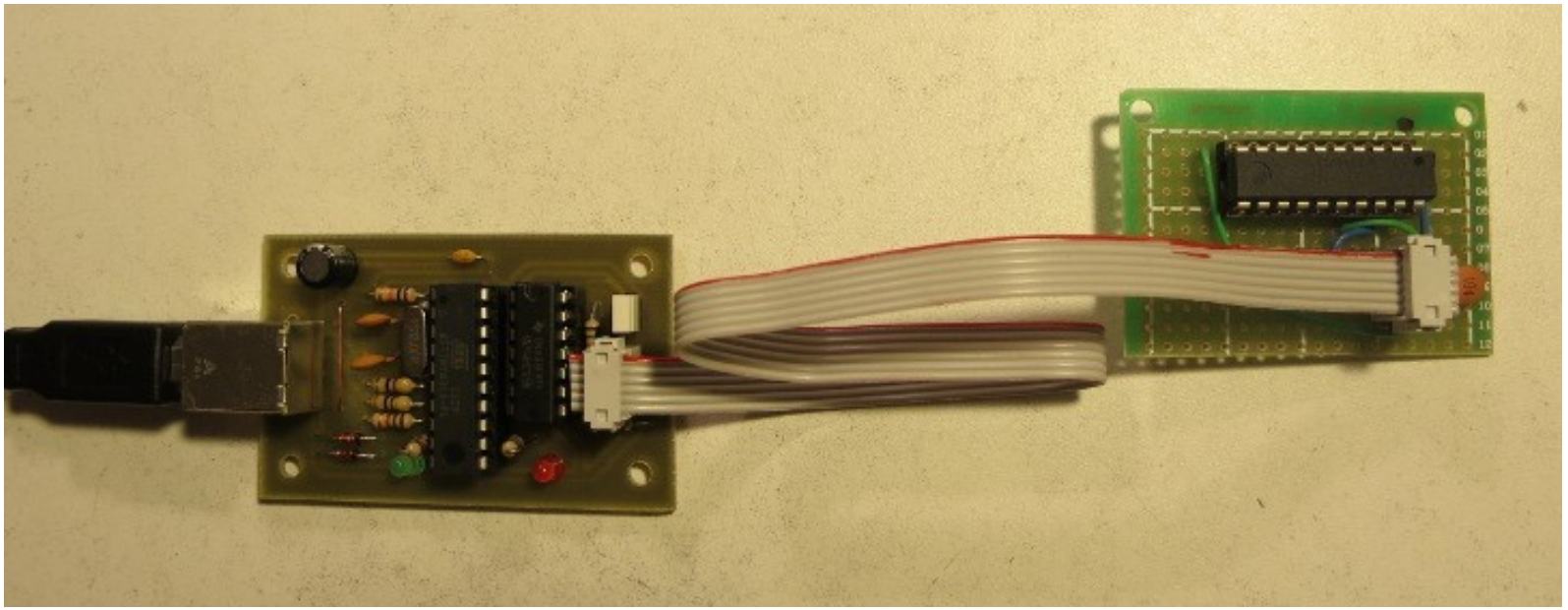
Workflow: Compiler

The preferred compiler for AVRs is avr-gcc. The input is your C code and the output is a .hex file, which contains the compiled firmware.

- Main page:
  - http://www.nongnu.org/avr-libc/
- Linux users have it easy:
  - sudo apt-get install gcc-avr binutils-avr avr-libc
- Windows users can play too:
  - http://winavr.sourceforge.net/
- Mac users might start here:
  - http://www.ladyada.net/learn/avr/setup-mac.html

Workflow: Connections

- Remove the MCU from your keypad
- Insert the MCU into the programmer break-out board (note pin 1!)
- Connect the 6-pin ribbon cable (note pin 1 on each end!)
- Connect the programmer to your computer



Workflow: Flash programmer

The USBtinyISP programmer works with the avrdude firmware uploader. This software reads in your .hex file and sends it to the flash programmer.

- Main page:
  - http://www.nongnu.org/avrdude/
- Linux users have it easy:
  - sudo apt-get install avrdude
- Windows users can play too:
  - http://winavr.sourceforge.net/ (avrdude seems to be included)
- Mac users might start here:
  - http://www.ladyada.net/learn/avr/setup-mac.html (there's some info here)

Dev kit

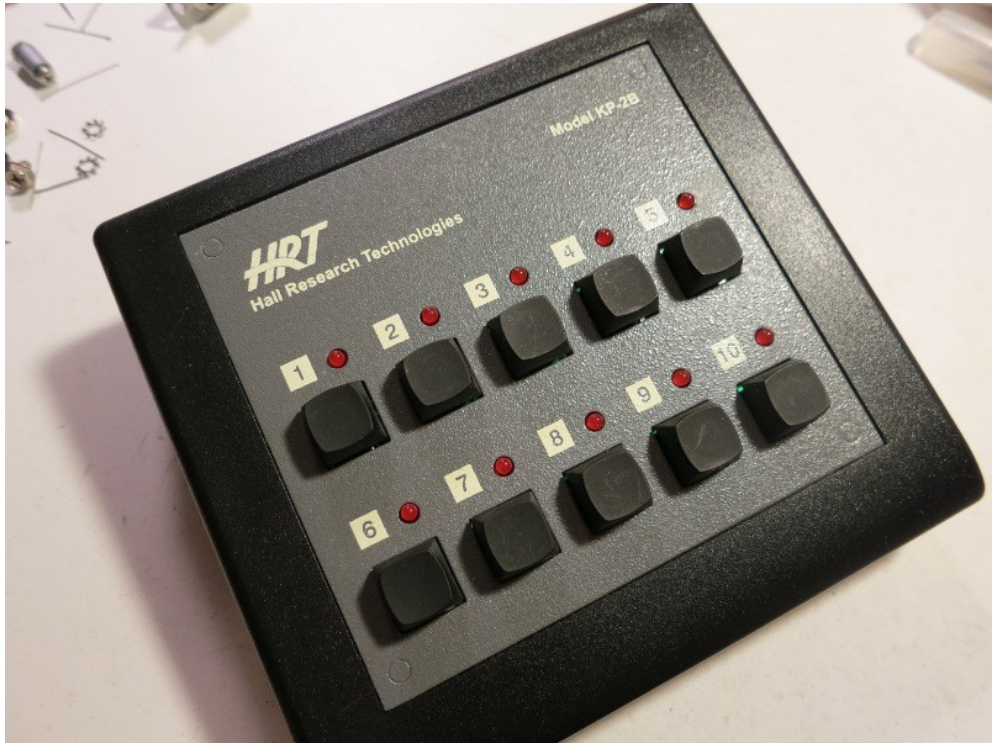
To help get people started in microcontroller development, I've put together some development kits. Here's what's included:

- Keypad
- Flash programmer
- “Target” board
- High tech power supply

Dev kit: The keypad

We found a bunch of these keypads in the trash. They've got a microcontroller, 10 buttons, 10 LEDs, and a serial interface. I've replaced the microcontrollers with Atmel ATtiny2313s.

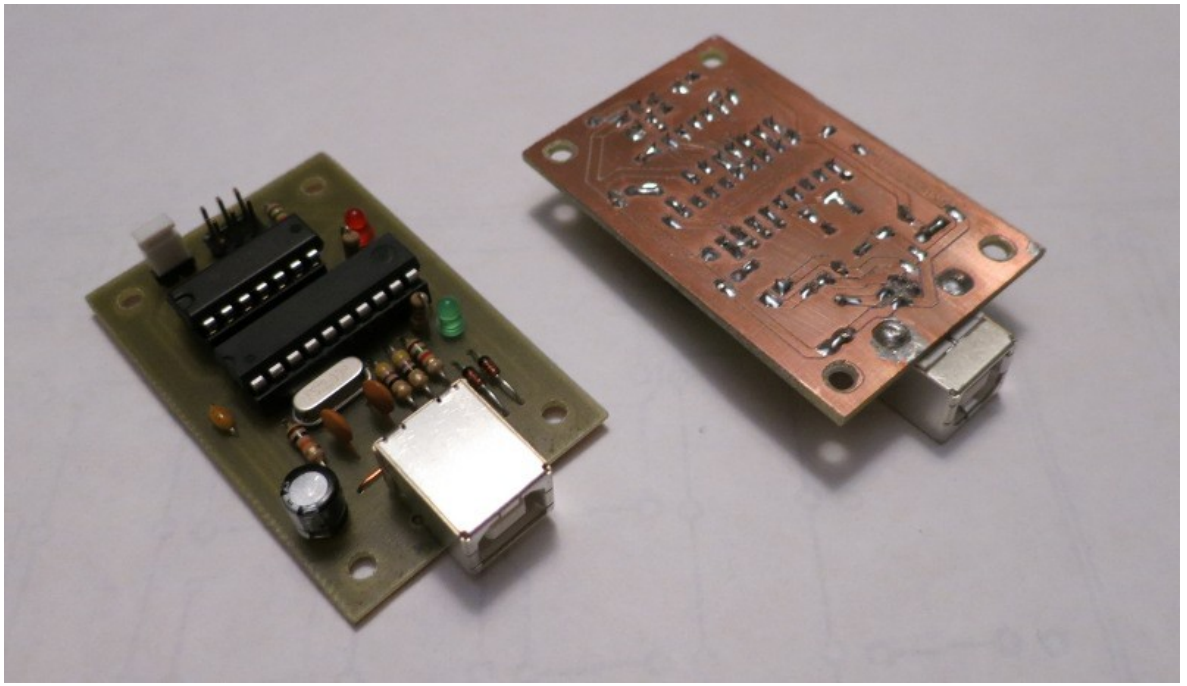
http://www.hallresearch.com/page/Products/KP-2B



Dev kit: The programmer

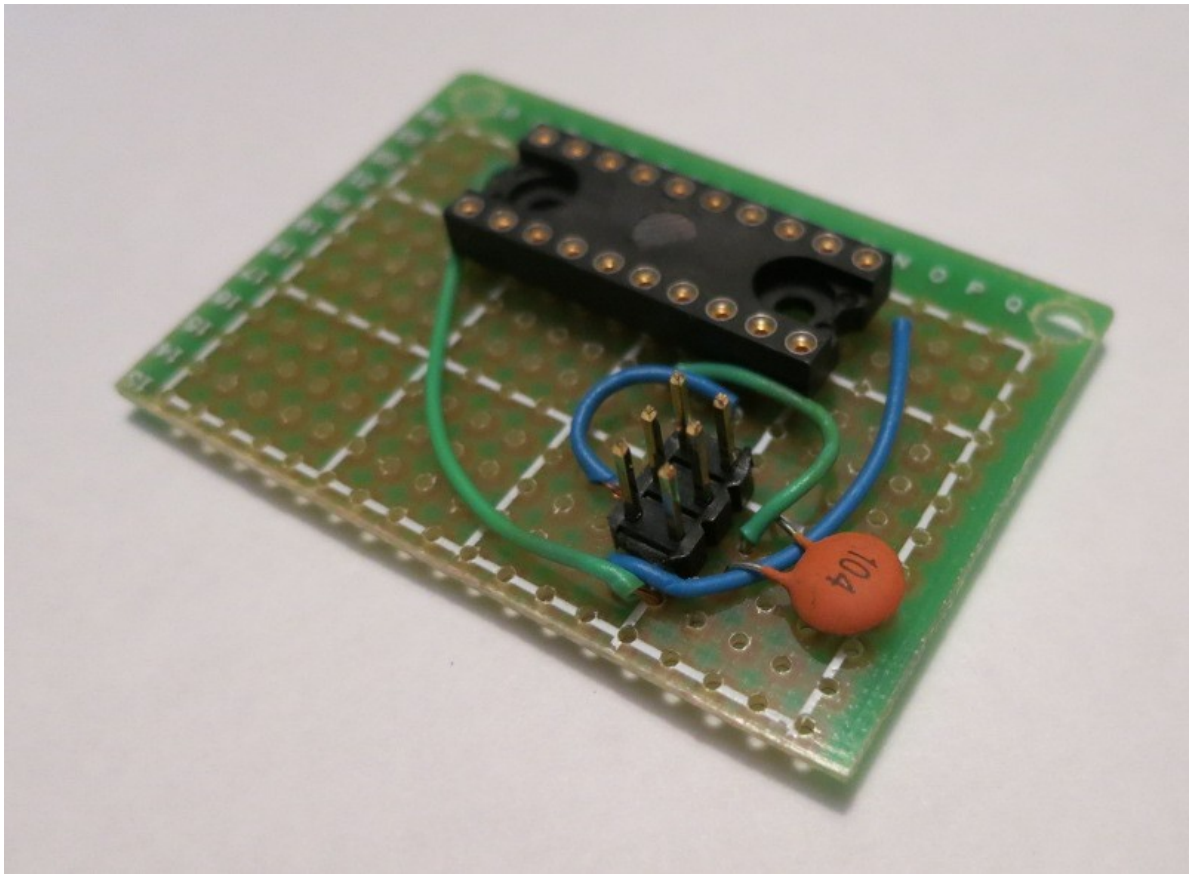
I made a bunch of flash programmers. They're based on the USBtinyISP. They allow you to write compiled software to the microcontroller.

http://www.ladyada.net/make/usbtinyisp/



Dev kit: The target board

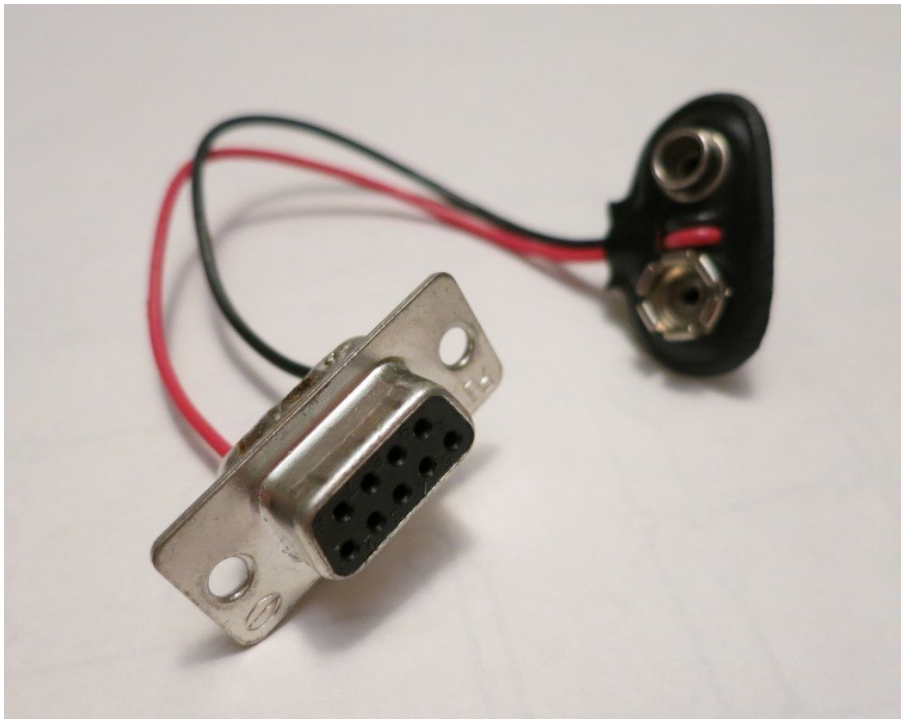
Nathan made a bunch of break-out boards. They connect your MCU to your flash programmer. Ordinarily, this feature is built in to the board you're developing for; in the case of the keypads, it is not built in.



Dev kit: High tech power supply

The keypad, in its original application, was powered via one of the data lines on the serial port. This space-age contraption allows you to power your keypad without tethering it to a computer.

Warning: Do not plug the high tech power supply into your computer or something. Doing so will surely create a black hole and destroy the Earth.



Shopping list

Here's what you need to complete your dev kit:

- USB A to B cable
  - This cable connects your programmer board to your computer.

Image courtesy of http://en.wikipedia.org/wiki/USB

Shopping list

- Chip puller
  - This tool aids extracting ICs from their sockets. Without one, your fingers will slip and you'll bend all the pins.

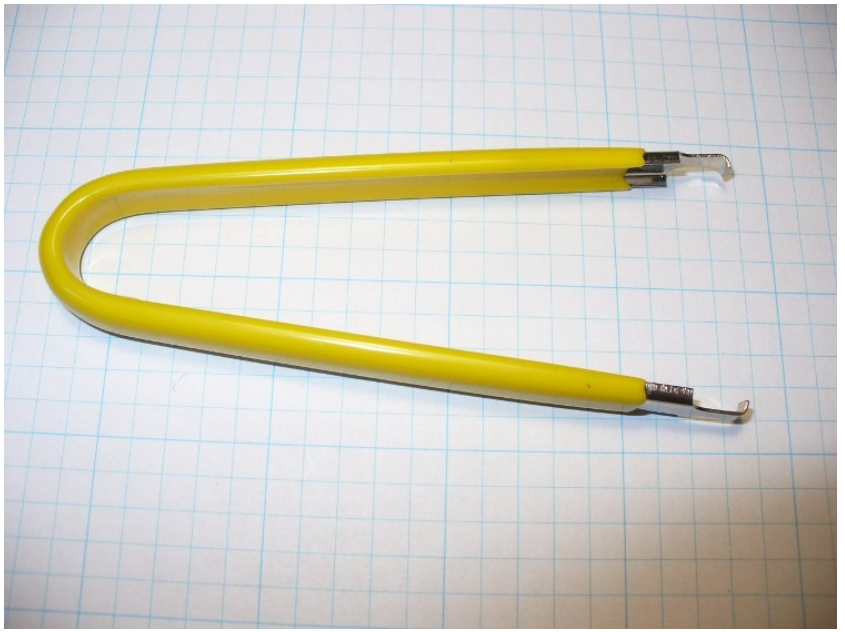


Image courtesy of http://tronixstuff.files.wordpress.com/2010/06/

Shopping list

- 9V battery
  - You need one of these in order to power your keypad.

Shopping list

- Optional: A case for your programmer
  - The programmer boards are vulnerable to corrosion from fingerprints and to metal bits shorting the pins on the bottom. You can buy cases for your programmers from here:
  - http://dangerousprototypes.com/docs/Sick_of_Beige_compa

Contest rules

- Write an interesting game for your keypad
- Entries must be open-sourced
- Deadline is January 14, 2013
- Submit your entries by forking my project on github and posting your code there.
- Prizes:
  - Everyone who submits an actual entry gets to keep the keypad and programmer. If you take a development kit today but don't produce anything by the deadline, I ask that you return the kit.
  - Grand prize: [TBD]
- Everyone who submits an actual entry gets to vote for the winner of the grand prize.

Example code

- I've reverse-engineered the keypad and written a demo app for it. The demo shows how to write to the LEDs and read from the switches.
- The example makefile contains examples of how to invoke gcc-avr and avrdude.
- The C files contain reusable functions for setting the LEDs and reading the switches.
- The project is hosted on GitHub. See http://github.com/andrewsampson
- Grab the code like so:
  - git clone [fixme]

Things I'm skipping

- Fuses
- The external crystal
- UART
- Timers
- EEPROM

Additional resources

Nathan offers the following additional AVR references.

- Basic:
  - Bit manipulation: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=37871
  - I/O operations: http://iamsuhasm.wordpress.com/tutsproj/avr-gcc-tutorial/
  - Storing constants in flash: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=38003
- Intermediate:
  - Efficient AVR C coding: http://www.atmel.com/Images/doc1497.pdf (syntax differences from GCC)
  - EEPROM: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=38417
  - Timers: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=50106
- Advanced?:
  - Interrupts: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=89843
  - Inline assembly: http://www.nongnu.org/avr-libc/user-manual/inline_asm.html
  - Multitasking: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=95490
- Misc/Resources:
  - ATtiny2313/4313 datasheet: http://www.atmel.com/images/doc8246.pdf
  - AVR forum with active community: http://www.avrfreaks.net/index.php?name=PNphpBB2&file=index
  - avr-libc FAQ: http://www.nongnu.org/avr-libc/user-manual/FAQ.html
  - RS232 tranceiver datasheet: http://www.datasheetcatalog.org/datasheet/sipex/SP3232.pdf
- Tips/gotchyas:
  - Use the smallest-sized data type that you need.
  - Don't change the fuses!
  - When changing fuses, don't change the reset disable fuse or the clock source fuses!
  - If using interrupts understand the concept of "atomicity" and when to use the keyword "volatile".
  - Don't use floating point math!
  - Don't divide!
  - If dividing by a constant, do it by multiplying by its reciprocal (google "binary scaling"--also used for multiplying by non-integer).
  - Need some extra speed? Look up "OSCCAL" in the attiny2313/4313 datasheet.