digitalWriteFast,pinModeFast digitalReadFast

One of the strengths of the Arduino platform is the low barrier to getting started for beginners. The simple syntax of digitalWrite, pinMode and digitalRead is a big contributor to the simplicity. One of the weaknesses though has been the relatively slow performance of those commands.

In the fall of 2009, Paul Stoffregen proposed and worked out the important details of a much faster version, which he implemented completely as a macro. It is over 20 times faster than the standard digitalWrite. The extra speed depends on the pin numbers (and HIGH/LOW values) being known at compile time--it won't speed things up if its inside a loop or subroutine where the pin number is going to change. If the pin number is not known at compile time it defaults to use the standard (slower) digitalWrite command. It uses the simple syntax of digitalWrite type commands which makes it attractive to beginners, and perhaps less error prone even to programmers who are a little more experienced.

I looked at what Paul had done and thought that with a little more work, it could be a valuable library 'routine'--I put routine in quotes because there's no .c file; everything is in macros in a .h file.

I extended it to include pinModeFast and (with a huge amount of assistance from Bill Westfield) digitalReadFast. The original version turned off PWM as have the standard commands. It was not however interrupt safe.

In the original version of this library I supplied 2 versions of each command with different behavior around turning off PWM (the analogWrite facility). This version does not turn off PWM--that feature is planned to be removed from future versions of the standard commands. In this version I only supply 1 version of the commands

Much more importantly this version is interrupt safe.

There are several good descriptions of the need for interrupt safe digitalWrite and pinMode commands. I will give yet another (not necessarily better) description at the end of this writeup.

To use digitalWriteFast, just put the digitalWriteFast folder into your library alongside the EEPROM, Ethernet, Firmata, etc folders. In your source code
#include <digitalWriteFast.h>
then you can use digitalWriteFast(pin,HIGH/LOW), digitalReadFast(pin), pinModeFast(pin,INPUT/OUTPUT) very much as you have used the built-in commands. The object code will not only be faster, but typically smaller as well.

The downside

Without the huge performance advantage motivating you to learn to use the PORT, DDR and PIN registers directly, you may not learn to take advantage of them. If you need to manipulate several adjacent pins at once you may be able to do it with one command and get another performance increase; learning to use those commands may also move you closer to learning the commands to get better control over the PWM pins for finer control of those.

Lastly, there is the likelihood that you will use one of these ‘Fast’ commands in a way that means the pin number is not known at compile time. This could mean that you think you’re getting high performance when you are really not. It might be hard to find that issue.

Thanks

Paul Stoffregen did the heavy lifting on this and deserves most of the kudos. I wouldn’t have made my small contribution without Bill Westfield’s patient and insightful coaching. Any problems are due to my shortcomings, not theirs.

Appendix

The need for interrupt safe pinMode and digitalWrite (often referred to as issue 146) has been described several times, perhaps better than I can. It struck me that I have a disassembly listing of the commands that certainly can be used to describe the situation in a different, not necessarily better, way:

For Ports F and below, a single instruction can be used to set or clear an individual bit that corresponds to the digitalWrite or PinMode setting you want to change. It will look like this:

```
pinModeFast(61,OUTPUT);
   390a:        87 9a         sbi     0x10, 7         ; 16  set a bit of port to 1
digitalWriteFast(61,LOW);
   390c:        8f 98         cbi     0x11, 7; 17      ; clear a bit of port to 0
```

The cbi and sbi instructions are inherently interrupt safe.

For Ports above F, the microprocessor must read in all the 8 pins in a Port, modify and rewrite all 8 pins. It would look like this if (it were done in an interrupt unsafe fashion):

```
digitalWriteFast(64,HIGH);
   38fe:        80 91 08 01   lds     r24, 0x0108     ;read the 8 bits of port into register 24
   3902:        84 60         ori     r24, 0x04       ; set a bit in register 24 to 1
   3904:        80 93 08 01   sts     0x0108, r24     ; write register 24 back to the port
```

The cbi and sbi instructions cannot address the ports higher than F.

The problem arises when the port is modified by other means in the microsecond between it being read into the register, modified and rewritten. Most commonly this seems to come up with the Servo library, but any code that has an interrupt routine that itself does a digitalWrite or pinMode might cause the issue.

What happens is that the 'lds   r24, 0x0108' would read the current information from the port. Then it might happen that an interrupt would divert the microprocessor to handle another task. Imagine that 0x108 is modified during processing of the interrupt. Then the microprocessor returns to your program, which proceeds with 'ori r24, 0x04' setting the bit you wanted, and 'sts   0x0108, r24' destroying whatever the interrupt routine did. The interrupt could come between the ori and the sts as well, of course. The problems are notoriously hard to debug; the exact timing of the interrupt cannot be replicated and the bits of code that interact are likely to be located far from each other--perhaps even in a library, whose source code you have never read!

This version of the library correctly generates interrupt safe code that looks like this:

```
digitalWriteFast(64,HIGH);
   38fa:        9f b7         in      r25, 0x3f       ; 63  read the interrupt status
   38fc:        f8 94         cli                     ; turn off interrupts
   38fe:        80 91 08 01   lds     r24, 0x0108
   3902:        84 60         ori     r24, 0x04       ; 4
   3904:        80 93 08 01   sts     0x0108, r24
   3908:        9f bf         out     0x3f, r25       ; 63  restore the interrupt status
```

This is needed for digitalWriteFast and pinModeFast. The standard digitalWrite was corrected to cover this situation in Arduino 19.