digitalWriteFast, digitalReadFast, pinModeFast etc.

One of the strengths of the Arduino is the low barrier to getting started for beginners. the simple syntax of digitalWrite, pinMode, and digitalRead is a big contributor to the simplicity. As beginners get more experience they move toward the more efficient port manipulation commands (see http://www.arduino.cc/en/Reference/PortManipulation, BUT some of the port information there is actually incorrect for the Mega, see http://spreadsheets.google.com/pub?key=rtHw_R6eVL140KS9_G8GPkA&gid=0). The port manipulation commands control the same pins but refer to them in a completely different syntax and depend on specific details of the pin being known at the time the program is written. It is difficult using port manipulation to mimic the simplicity of

```
for (int i =2; i<=13; i++) digitalWrite(i,HIGH);
```

A few months ago, Paul Stoffregen proposed and worked out the important details of a somewhat intermediate version digitalWriteFast which he implemented completely as a macro. Like the port manipulation commands it is much faster than digitalWrite (over ten times faster) and the extra speed depends on knowing the pin numbers at compile time--it won't speed things up if its inside a subroutine or loop where the pin number is going to change. If the pin number is not known at compile time, it defaults to use the slower digitalWrite command. It uses the simple syntax of the 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 he had done and thought it needed just a little more work to make it 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 Westfw) digitalReadFast. I've tested it fairly thoroughly on my Arduino Mega. It would be wise if people with other Arduinos would test those boards. Paul Stoffregen's work on defining the port and bit to pin conversion has been flawless but testing seems prudent.

PWM, analogWrite digitalWriteFast2, etc.

As you know analogWrite works on some of the digital pins (the PWM pins by setting a timer and cycling from +5V to ground with a duty cycle that makes the average voltage on the pin proportional to what you specify. The standard digitalRead and digitalWrite commands turn off the cycling of the timer every time they are used. pinMode does nothing to the cycling of the timer.

digitalWriteFast and digitalReadFast turn off the cycling of the timer every time they are used. pinModeFast does nothing to the cycling of the timer. This is the mode for maximum compatibility.

However there is a comment in the code for the standard commands that suggests it would be more efficient to turn off the timer in pinMode and not in the other 2

commands. This makes enormous sense to me; in many instances pinMode might be used just in setup() and so the extra overhead could be dispensed with.

I was reluctant to completely split with the maximum compatibility that more experienced developers had opted for. So I did it both ways. digitalWriteFast2 and digitalReadFast2 don't turn off the timer. pinModeFast2 does turn off the timer.

What's in the download
A folder called digitalWriteFast, containing digitalWriteFast.h and a keyword file. That goes in your library. A program called digitalWriteFastTest.pde which is what I used to test it on my mega. If you want to test it for a smaller board, you should be able to delete a huge number of test cases. There is also a program called progprog.py; because these new commands deliver their speedup when the pin numbers are known at compile time I needed to test the commands with source code that specified the pin numbers directly. With 50-some pins to test I needed to generate most of the test program source code automatically. progprog.py generates that test code. You would need something like it if you want to test on a seeduino Mega, for example.

To use it 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)`, `pinMode(pin,INPUT/OUTPUT)`, `digitalWriteFast2(pin,HIGH/LOW)`, `digitalReadFast2(pin)`, `pinMode2(pin,INPUT/OUTPUT)` very much as you have used the built-in commands. The object code will not only be faster, but actually 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 my small contribution without Westfw's patient and insightful coaching. Any problems are due to my shortcomings, not theirs.