Modifications to LiquidCrystal for the Arduino with callback busy test

This version of LiquidCrystal supports the API of LiquidCrystal in Arduino 17 fully but does not actually ever use 8 data pins. This change makes the code smaller, saving flash RAM and also less complex, making it easier to read.

40x4 LCDs
I added support for an LCD of 4 LInes and 40 characters. I think that if 24x4, 32x4 LCDs exist, they would also work with the software as I have modified it although I have not had the opportunity to test that. The 40x4 LCD (and any HD44780 based LCD with between 81 and 160 characters) will have 2 enable lines. To use an LCD with 4 lines and 40 columns you would declare your LiquidCrystal object as:
LiquidCrystal lcd(RS,RW,Enable1,Enable2, data3,data2,data1,data0); at this time I don't support 8 data lines. (You can pass 255 as the RW item, ground RW and save an Arduino pin.)
Then in the setup function you would call:
lcd.begin(40,4);

Linewrap
When you declare the dimensions of the LCD in your begin call, the LiquidCrystal library remembers how long the lines are. Now when it reaches the end of line 1, text wraps onto line 2 (not line 3 as previously).

println
Although print has worked properly in the past, println has not. Now the '\r' and '\n' characters are not sent to the screen as though they were visible characters and the '\r' resets the character position to the top of the next line.

16x4 LCDs
The begin statement also correctly positions text at the beginning of the line on 16x4 (and 40x4) LCDs, which were not correctly handled before.

setCursor
In the past setCursor selected a location in the HD44780's RAM not actually a screen location. If you use any of the commands that shift the display left or right with the previous routines, then setCursor and print, text appears in an unexpected location on the screen. With the new software, if you call either scrollDisplayLeft() or scrollDisplayRight(), the LiquidCrystal package keeps track of the relationship between RAM and the LCD so that setCursor coordinates are pegged to a specific spot on the screen, rather than a spot in RAM. The sotware does not handle autoScroll, however. Call home() after autoScroll to restore the expected relationship between setCursor and the LCD screen.

Testing the LCD Busy Flag
Previous versions of LiquidCrystal always used timed delays on the Arduino side of

the interface to give the LCD module enough time to complete its operation. This version still does that if you tie the RW pin to ground and do not tell LiquidCrystal what that pin number is or pass it the address of a user routine to test the busy flag. If you do specify RW now, however, the software will poll the busy flag on the LCD module. Arduino operations may thus overlap LCD operations and potentially things may go a little faster.

Syntactic Sugar

#include <Streaming.h> from http://arduiniana.org/2009/04/new-streaming-library/
Then you can combine that with an overloading of the () operator in this code. This lets you specify screen location and chain print commands together by writing:
lcd(column,line)<<"a="<<a;
Streaming.h is so efficient you may actually save a few bytes of memory!

Speed testing

All of the interface modes go faster than the eye can follow. This version of the software is even slower than the previous version when using timed delays. I found that my LCD (Axman) needed an even longer delay than before. In the interests of making the code foolproof, I lengthened the delays to make that LCD work. However Paul Stoffregen has significantly speeded up the code when testing the busy flag and so those options run significantly faster than before. I compared the speeds of the different interfaces--writing 80 characters to the screen then 80 blanks and looping through that 20 times. The results on a Mega are:

```
Axman 4 data pins no RW 1417 milliseconds  |  nonAxman 1417
Axman 4 data pins  + RW  565 milliseconds  |  nonAxman  468
Axman 4 pins + user busy 369 milliseconds  |  nonAxman  316
```

I also have a Teensy++2.0 board. One of the interesting things about that board is that the software that comes with it includes considerable optimization of digitalRead, digitalWrite etc. The board runs at 16 megaHz, just like the Mega, but speeding up those commands results in an impressive change in the benchmarks:

```
Axman 4 data pins no RW 1276 milliseconds  |  nonAxman 1276
Axman 4 data pins  + RW  330 milliseconds  |  nonAxman  222
Axman 4 pins + user busy 241 milliseconds  |  nonAxman  190
```

Crazy 8 Addressing

16x1 LCDs often have an unusual address layout; these modules often have two 8 character halves and work best with this software if you declare them as lcd.begin(8,2); if you do that, then you can print("abcdefghilklmno"); and have all the characters appear as you would like across the screen. If you use any of the scrolling commands, the bizarre addressing of these modules will manifest itself. For details follow the _LCD Addressing_ link at web.alfredstate.edu/weimandn

User callback busy test
Get LiquidCrystal running without this first; setting up this complicated option is error-prone and this should be left for last, if implemented at all.

The LCD has a busy flag which is intended to notify the processor when it is ready to accept a new command. Testing it involves manipulating almost all the pins in the interface and is done somewhat slowly inside the library routine because of the overhead of pinMode,digitalRead and digitalWrite. If the pin and ports were always the same, faster manipulation would be possible. In fact, the port to pin correspondence hasn't remained constant across boards, so even if everyone used the same pin numbers, code that manipulated ports would fail:

Digital Pin	Arduino 8 Port	Arduino 168/328 Port	Arduino Mega Port
0	PD0	PD 0	PE 0
1	PD1	PD 1	PE 1
2	PD2	PD 2	PE 4
3	PD3	PD 3	PE 5
4	PD4	PD 4	PG 5
5	PD5	PD 5	PE 3
6	PD6	PD 6	PH 3
7	PD7	PD 7	PH 4
8	PB0	PB 0	PH 5
9	PB1	PB 1	PH 6
10	PB2	PB 2	PB 4
11	PB3	PB 3	PB 5
12	PB4	PB 4	PB 6
13	PB5	PB 5	PB 7

A faster way to be able to test the busy flag and know the pin/port numbers is to use a callback function and let the testing be done in the user's code rather than inside the library.

if you've ever used attachInterrupt on the Arduino, you've used a callback function. You write a function and pass the address of the function to another routine in a function call. In this case, I've added a couple more forms of the (very overloaded) LiquidCrystal object. Now you can specify your callback function like this for the common case of displays up to 20x4:

```
LiquidCrystal lcd(49,45, 35,33,31,29,&checkBusyFlag);//RS,EN,D0,D2,D1,D3
```

Notice that in this case, RW must be attached to a digital output pin and the user code must set that pin LOW before calling LiquidCrystal. The busy routine will be manipulating the status of RW but the main LiquidCrystal routine doesn't manipulate it or know which pin it is. (These pin numbers will likely look unfamiliar to you; on my Mega, I like to plug the LCD into the socket on the board directly with pin 1 in a

ground, pin 2 into socket 53, pin 3 in socket 51 and so on.)

You would declare your LiquidCrystal object like this for a 40x4 LCD :
`LiquidCrystal lcd(48,255,46,52, 41,40,39,38,&checkBusyFlag);//RS,RW,EN,EN2,D0-D3`
You will have to make the RW pin LOW before you call lcd.begin(40,4).

Then you can modify the macro definitions which are immediately below (busy is the data3 item in your call to LiquidCrystal):

```
//LiquidCrystal lcd(49,45, 35,33,31,29,&checkBusyFlag)  //RW is pin 47
#define set_data_pins_to_read DDRC&=~0b01010100,DDRA&=~0x80;//d0,d1,d2,d3 pins35,33,31,29
#define set_data_pins_to_write DDRCl=0b01010100,DDRAl=0x80;    // C2,C4,C6,A7
#define set_EN_high PORTL l= 0b00010000;    //port L4 pin45
#define set_EN_low PORTL &= ~(1<<4);
#define set_EN2_high PORTL l= (1<<4);    //portL4 the EN2 if you have a 40x4 LCD
#define set_EN2_low PORTL &= ~(1<<4);
#define set_RW_high PORTL l= (1<<2);
#define set_RW_low PORTL &= ~(1<<2);    //port L bit 2 pin47
#define set_RS_high PORTL l= (1<<0);    //port L bit 0  pin 49
#define set_RS_low PORTL &= ~(1<<0);
#define read_busy PINA & (1<<7);        //port A7  pin29   the last pin number spec'd to LiquidCrystal

//You DON'T have to modify the subroutine itself:
void checkBusyFlag(int8_t chip) {
 uint8_t busy; // = 0x04;
 set_data_pins_to_read;
 set_RW_high;            //RW to read
 set_RS_low;
 if (chip == 0) {  //the if and else can be eliminated if only one hd44780 chip eg 20x4
 do {
   set_EN_high;
   delayMicroseconds(1);
   busy = read_busy;    // read busy flag
   set_EN_low;
   delayMicroseconds(1);
   set_EN_high;
   delayMicroseconds(1);  //pulse the second nibble--discard it;
   set_EN_low;
 }while (busy);
} else {
   do {
   set_EN2_high;
   delayMicroseconds(1);
   busy = read_busy;    // read busy flag
   set_EN2_low;
   delayMicroseconds(1);
   set_EN2_high;
   delayMicroseconds(1);  //pulse the second nibble--discard it;
   set_EN2_low;
 } while (busy);
}
```

```
 set_data_pins_to_write; // data pins to write
 set_RW_low;             //RW to write
 set_RS_high;
}
```

Again, this is complicated and you can expect to make some mistakes in the define statements at first. The payoff is faster operation than with the 4 data pin option. It does cost you one Arduino pin for RW. I have not created 8 data pin versions of these.

Disadvantages

The only real disadvantage I can see to the changes I have made is the possibility that someone with a little 16x2 LCD is using the scrollDisplayLeft() or scrollDisplayRight() instructions to move data across the screen, but wants to write the data 40 characters at a time with a print statement. This version really does not let the user write data to the HD44780 DDRAM which is not visible. To accomplish a scrolling display with 40 characters per line, you would now need to write 16 characters, scroll the display, write a little more and so on.

There are going to be some incompatibilities between code that assumed that line 1 wrapped onto line 3, etc.

Directions for the future

Paul Stoffregen suggested making the internal function send() public rather than private so that classes that inherit from LiquidCrystal can provide an alternate function, presumably hard coding pins and getting still more speed. I made all of the 'private' functions and instance variables in LiquidCrystal 'protected'. I think this allows the same functionality, but have not yet tested this.

Bug

The one bug I am aware of is that autoscroll() on a 40x4 LCD only scrolls 2 lines despite my best efforts.

Thanks

Certainly my efforts would not have been possible without the help and prior efforts of David Mellis, Limor Friede, and Donald Weiman. Don was particularly patient in guiding me through the idiosyncracies of the HD44780 based LCDs and especially in supplying an example of how the busy flag could be tested. Paul Stoffregen contributed significant optimization of the code to make it faster.

Appendix: A more common pin configuration for the busy test

The more common pin usage for interfacing an LCD is probably that given at
http://www.arduino.cc/en/Tutorial/LiquidCrystal
to use the busy test, you will have to connect RW, in this example, to pin 10. To achieve enough board independence for me to test this, download digitalWriteFast (see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1267553811/0); it will take care of the pin to port issues without ALL of the speed disadvantages of digitalWrite,

etc. I can test this code on my mega, whereas I can't test the equivalent port version. Use this code, then:

```
#include <digitalWriteFast.h>
pinModeFast(10, OUTPUT);
digitalWriteFast(10,LOW);
LiquidCrystal lcd(12,11, 5,4,3,2,&checkBusyFlag); //pins from tutorial

#define set_data_pins_to_read
pinModeFast(5,INPUT),pinModeFast(4,INPUT),pinModeFast(3,INPUT),pinModeFast
(2,INPUT);   //data pins 5,4,3,2
#define set_data_pins_to_write
pinModeFast(5,OUTPUT), pinModeFast(4,OUTPUT), pinModeFast(3,OUTPUT),
pinModeFast(2,OUTPUT); // digitalWritefast etc gives some board independence
#define set_EN_high digitalWriteFast(11,HIGH);    //port B3 pin 11
#define set_EN_low digitalWriteFast(11,LOW);
#define set_EN2_high digitalWriteFast(11,HIGH);    //port B3 pin 11
#define set_EN2_low digitalWriteFast(11,LOW);
#define set_RW_high digitalWriteFast(10,HIGH);
#define set_RW_low digitalWriteFast(10,LOW);    //port B bit 2 pin10
#define set_RS_high digitalWriteFast(12,HIGH);    //port B bit 4  pin 12
#define set_RS_low digitalWriteFast(12,LOW);
#define read_busy digitalReadFast(2);        //portD2, pin 2
```

Then copy checkBusyFlag as listed above.