You may not have noticed, but each time you plug the Arduino in, the last program uploaded will start executing. The program that is moved to the Ardunio is placed in what is known as flash memory. This memory on the computer can be overwritten but is permanent even when the power is cycled. This means that if you want to put the Arduino in a piece of hardware and have it start up with your program running, all that you do is upload the program and thats it.

One problem is that most programs have some data that can be changed by the user, but you might want it to be held through a power off, power on cycle. For this very reason, Arduino has an Electrically Erasable Programmable Read Only Memory (EEPROM) as part of its hardware. Now this memory is rather small, 1024 bytes for the nano, and can only be written to a finite number of times, 1,000,000 in some of the documentation. However if you want to save off some parameter that the user has entered the EEPROM is the way to do it.

Problems that will need to be dealt with when writting to the EEPROM. First, don't write fast changing data to the EEPROM, keep primarily parameters, and only write them if there is a change. Second, when you are writing to the EEPROM, note it will take about 3.3 milliseconds to write a byte, thus long arrays can take a lot of time. Third, the EEPROM can only accept bytes (0 to 255) when writing data. Let us consider how we can address each of these problems in the easiest fashion.

Limited write cycle: Although 1000000 sounds like a lot, if the EEPROM is being written to every 1 second, this is 11 days. Again this sounds like a lot, but over the course of a year your system might actually run for 11 days. Fortunately the EEPROM software on Arduino has an update write, that first checks to see it the data being written is the same as the data in the EEPROM, and if the match it doesn't write, saving a write cycle.

As to the time required for an EEPROM write, little can be done, except to only write one word at a time. In other words if you had an array of 30 bytes to be written, that would take about 100 milliseconds. So we should only write each time through the loop. So if you have a large amount to write to EEPROM, consider having an array of bytes and an index such as

```
int EEP_ToWrite = 0;
unsigned char EEP_Data[40];

void loop()
{

  // fill EEP_Data with the data to be written
  // and then set EEP_ToWrite to the number of entries.

  ...

   if( EEP_ToWrite != 0 )
   {
          EEP_ToWrite--;  // move to next data to write.
          // Write this word if it has changed.
          EEPROM.update( EEP_ToWrite, EEP_Data[EEP_ToWrite] );
    } // End of EEP write
```

To reiterate that it really does take time to write to the EEPROM, the code in Appendix A was written and run on the Arduino Nano. Figure EEP-1 shows that 3.45 seconds were required to write the full EEPROM of 1024 words. However the update was faster, mostly because it didn't have to write to any of the locations. Finally reading is basically the same as accessing data in memory.

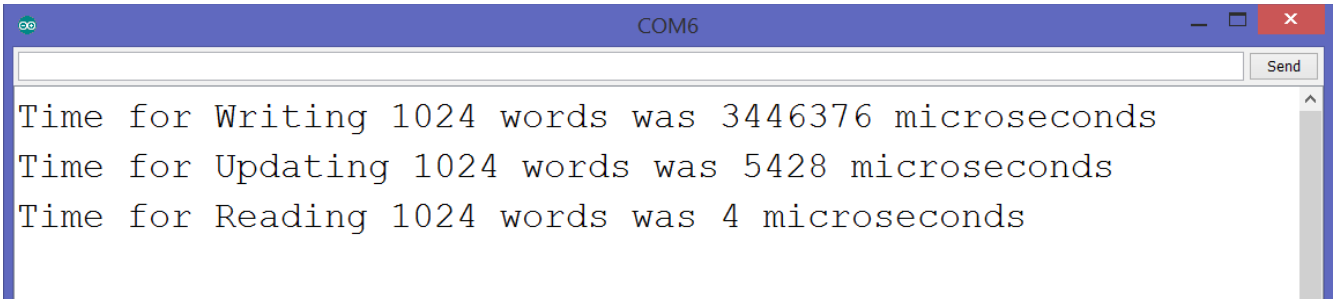

Figure EEP-1. Results of EEPROM Read and Write Timing Test.

Now often you may have numbers that are not easily broken into bytes, such as int's or float's. In the case of an int, there are two functions in the Arduino environment called lowByte( ) and highByte(). These functions can be called in succession to pull out the two bytes for writing, such as

```
EEPROM.write( address, highByte( value ) );
EEPROM.write( address+1, lowByte( value ) );
```

To read the data back, we would do something like

```
value = EEPROM[ address ] * 256 + EEPROM[address+1];
```

However the more complex structure of a float doesn't allow it to be easily split into bytes. To do this effectively we can use an unusual construct in c called a union. I used one to write two temperatures in degrees Fahrenheit to EEPROM is my smoker controller in the following fashion. The union overlays the floating point number and the array of four bytes. Then the bits can be accessed as a floating point number (LowTemp.F), or each byte can be accessed separately (LowTemp.B[i]).

```
// Declared as globals.
union flt_byte { float F; unsigned char B[4]; } LowTemp, HighTemp;
int WriteTempData = -1;
...

void setup()
{

   LowTemp.B[0] = EEPROM[0];
   LowTemp.B[1] = EEPROM[1];
   LowTemp.B[2] = EEPROM[2];
   LowTemp.B[3] = EEPROM[3];

   HighTemp.B[0] = EEPROM[4];
   HighTemp.B[1] = EEPROM[5];
   HighTemp.B[2] = EEPROM[6];
   HighTemp.B[3] = EEPROM[7];

}

void loop()
{

     if( WriteTempData > 3 )
     {
              EEPROM.update( WriteTempData, HighTemp.B[WriteTempData-4]
              WriteTempData--;
      }
      else if( WriteTempData > -1 )
      {
              EEPROM.update( WriteTempData, LowTemp.B[WriteTempData]
              WriteTempData--;
      }

      ...
      // Access to the floating point value is done by LowTemp.F
      // if values change and need to be written to EEPROM, simply
      WriteTempData = 7;
```

Appendix A: Arduino code testing write time.

```
#include <EEPROM.h>
// Program that only runs through and writes to the EEPROM,
// Timing how long it takes.
void setup()
{
	int i, j; // generic indicies
	unsigned long Start,	// These are used to
		WriteTime, // measure time via
		UpdateTime,// the micros function.
		ReadTime;

	Serial.begin(9600); // Set up serial port,

	// First step is to write 1024 words to the
	// EEPROM, measuring how long it takes.
	Start = micros();
	// Loop through EEPROM filling it
	for (i = 0; i<EEPROM.length(); i++)
	{
		// Write EEPROM.  Note i will be truncated to
		// 8 bits or 0 to 255.
		EEPROM.write(i, i);
		//EEPROM[i] = i; // Alternate way to write,
		// takes just as long.
	}
	WriteTime = micros() - Start;  // Compute time to write.

	// Now loop through doing an update, instead of write.
	Start = micros(); // Snap shot of time.
	// Loop through EEPROM updating each location.
	for (i = 0; i<EEPROM.length(); i++)
	{
		EEPROM.update(i, i);
		// Only writes if value different than
		// what is already in EEPROM.
	}
	UpdateTime = micros() - Start; // Compute time.

	// Now test how long it takes to read data.
	Start = micros(); // Starting time.
	// Loop through EEPROM reading each byte.
	for (i = 0; i<EEPROM.length(); i++)
	{
		j = EEPROM[i];
	}
	ReadTime = micros() - Start; // Compute time
```

```
        // Send results to terminal.
        Serial.print("Time for Writing ");
        Serial.print(EEPROM.length());
        Serial.print(" words was ");
        Serial.print(WriteTime);
        Serial.println(" microseconds");

        Serial.print("Time for Updating ");
        Serial.print(EEPROM.length());
        Serial.print(" words was ");
        Serial.print(UpdateTime);
        Serial.println(" microseconds");

        Serial.print("Time for Reading ");
        Serial.print(EEPROM.length());
        Serial.print(" words was ");
        Serial.print(ReadTime);
        Serial.println(" microseconds");

}

void loop()
{
        // Empty since nothing to do.

}
```