# Adding External I2C EEPROM to Arduino (24LC256)

## Adding External I2C EEPROM to Arduino (24LC256)

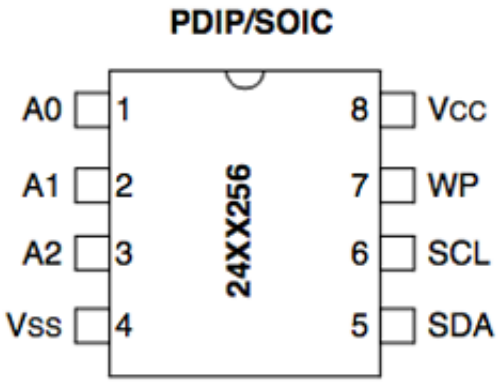
This tutorial was originally posted on the 10kohms.com website, which now seems to be no longer with us, so we have reproduced it here.

In my last post I discussed using the built in EEPROM to store permanent data on the Arduino. All though this is a very easy and effective way of storing data on the Arduino the built in EEPROM only offers 512 bytes of storage. When working with larger or more advanced Arduino projects we may need to store additional data so an external memory solution like the 24LC256 I²C EEPROM IC becomes necessary.

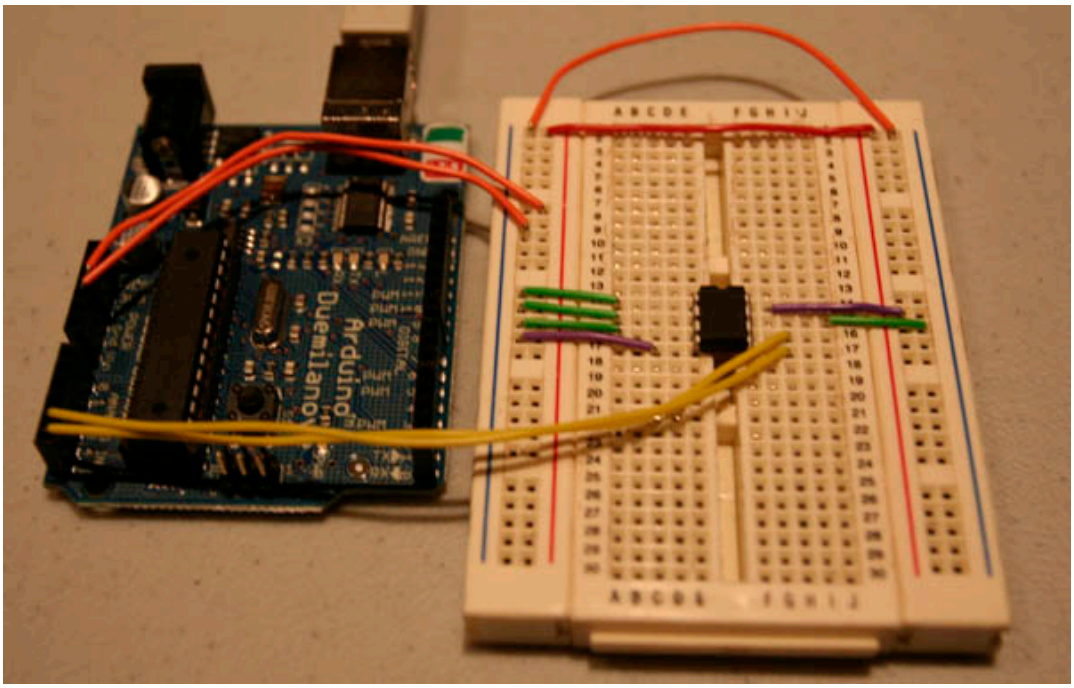
We're using a 256kbit eeprom which is actually 32kbytes of space. 262,144 bits / 8 bits in a byte = 32,768 bytes. That's 62 times the Arduino's built-in storage!

**Hardware Setup**

In this example we'll be using the Microchip 24LC256 IC. If you're using a different IC please confirm that the pin-out and power requirements are the same so you don't damage your chip.



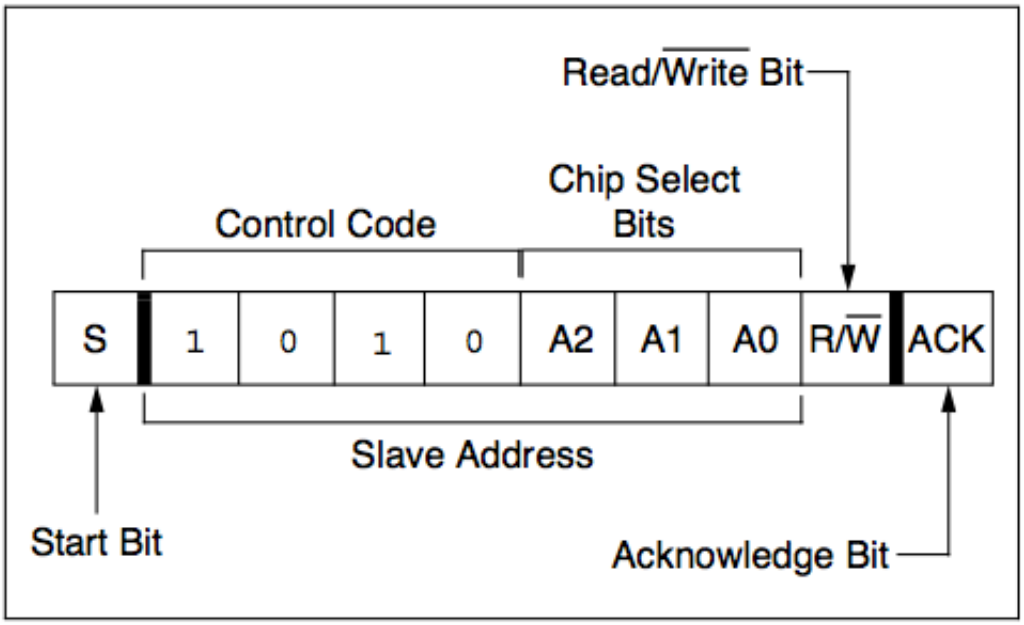

The Microchip 24LC256 chip can be purchased in a 8 pin DIP package. The pins on the 24LC256 are pretty straightforward and consist of power(8), gnd(4), write protection(7), SCL/SDA(6,5), and three address pins(1,2,3). Before we get into the software part lets hook up the 24LC256 chip up to our Arduino.



Using the image above as a guide lets begin to wire the chip. First connect GND and VCC, pins 4 and 8 respectivly. Next lets go ahead and connect the data pins to the Arduino board. Since we're using the Arduino I²C bus we're going to be using **Analog** pins 4 and 5. Connect the SDA pin on the 24LC256(pin 5) to the pin 4 of the Arduino. Then connect the SCL(pin 6) to pin 5 on the Arduino. Double check that you've connected the correct pins on the 24LC256 to the correct pins on the Arduino; strange things will happen if

you have them reversed. After our data and power pins are connected we have four left on 24LC256 chip, the WP pin and the three address pins. The WP pin stands for write-protected and this allows you to control if data can be written to the eeprom or not. If this pin is low then writing is enabled but if it's high then writing is disable; reading is always enabled. For the purpose of this tutorial we're going to be writing to the eeprom so we can connect the WP pin to GND.

The last three pins set the address of the 24LC256 chip which allows us to target a particular chip on the I²C bus. This particular I²C chip comes pre-wired with four bits of it's address already set(1010) and these can not be changed. The last three bits of the address however can be changed which allows us to run up to eight 24LC256 chips on the same I²C bus. This is a little confusing at first so lets look at the figure below to explain the address in a little more detail.




For the purpose of explaining how the address works we can ignore the Start and Acknowledge bits. The way the I²C bus works is a 7-bit address is passed along with a read/write bit that tells the chip if it should write the incoming data or read it and send it back. The Arduino takes care of the last R/W bit for us depending on what function we're using so as long as you're using the standard Arduino Wire library we don't have to worry about this bit. This

leaves us with the seven middle bits and as I mentioned above the first four bits(Control Code) are hard-wired and we can't change these. The next three bits(A2,A1,A0) are the important bits that we can change so lets look at the simple table below to see what address the chip will have depending on what we set these pins to.

So, if we were to tie pins 1,2 and 3 on the 24LC256 to GND then the chip would have address 0×50 and if were to assign them all Vcc then the chip would have address 0×57 and every combination in between. To keep things simple lets just tie all pins to GND to make the address 0×50. In a future tutorial I will show you how to use multiple eeprom chips off the same I²C at which point we will be assigning each chip a different address but for now lets stick with 0×50. With the address pins connected the hardware part of this tutorial is complete and every pin of the 24LC256 should be connected to either Vcc, GND or the Arduino. Time to move on to software!

It's been brought to my attention that some people use pull-up resistors on the data and clock pins to the Arduino. All though this would not hurt the circuit it's not needed because when the Wire.h library is initialized it knows pins 4 and 5 are going to be used for I²C so it also activates the built-in pull-up resistors. For more information please read http://www.arduino.cc/en/Reference/Wire.

**Arduino Sketch**

Below is the entire tutorial code, scan over it and see if you understand it before I dive into what each section does.

Note: This is written for Arduino versions before 1.0. If you are using Arduino 1.0 and above then you need to change **Wire.send** to **Wire.write** and **Wire.receive** to **Wire.read**

```
#include <Wire.h>
```

```
#define disk1 0x50    //Address of 24LC256 eeprom chip

void setup(void)
{
  Serial.begin(9600);
  Wire.begin();

  unsigned int address = 0;

  writeEEPROM(disk1, address, 123);
  Serial.print(readEEPROM(disk1, address), DEC);
}

void loop(){}

void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
{
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.send(data);
  Wire.endTransmission();

  delay(5);
}

byte readEEPROM(int deviceaddress, unsigned int eeaddress )
{
  byte rdata = 0xFF;

  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.endTransmission();

  Wire.requestFrom(deviceaddress,1);

  if (Wire.available()) rdata = Wire.receive();

  return rdata;
}
```

In order to use the I²C interface we need to include the Arduino standard

Wire library so first things first, include Wire.h at the top of the sketch. You'll notice directly after the include we define a variable called disk1 and assign it a hex value of 0×50. When we setup the chip above we set the address of the chip to 0×50 by tying all the address pins to GNS so we simple set this variable to that address which allows us to access the chip from within our sketch. This variable is not required but it allows us to easily change the address we want to access without going through all of the code and replacing the value. Also, if you plan on adding more than one chip it's easier to refer to them as disk1, disk2, etc rather than 0×50, 0×51 which might get confusing.

Moving on we have our standard setup and a loop functions, for this tutorial the loop function is left empty so we'll just focus on the setup function. We first initialize our Serial connection for printing back to the computer and then we initiate the I²C connection by calling Wire.begin(). This enables pins 4 and 5 for I²C and also enabled the internal pull-up resistor(See note above). Next we create a new variable to store the address of the eeprom we want to write to(not the address of the eeprom IC itself but the address of the byte we want to read/write to). Since this eeprom has 32Kbytes of storage this address can be any number between 0 and 32,767; we'll start with address 0. After we've initialized everything we call our two primary functions, writeEEPROM and readEEPROM which actually do the dirty work of writing and reading the bytes of data.

Lets first start off with the writeEEPROM function. This function takes three arguments, the device address(the disk1 variable), the memory address on the eeprom and the byte of data you want to write. The first argument is the address of the device you want to write to, in our case we only have one device(disk1) so we pass this on. The next argument is the address on the eeprom you want to write to and as stated above can be between 0 and 32,767. Finally we have to pass along the byte we want to store. So, writeEEPROM(disk1, address, 123) is going to write the decimal 123 to "address"(which is 0) on disk1(0×50). Lets jump into the actual writeEEPROM function to learn what it does.

```
void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
{
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.send(data);
  Wire.endTransmission();

  delay(5);
}
```

We first call the Wire.beginTransmission function which sends the deviceaddress to let the chip know we want to communicate with it. Next we have to send the address on the eeprom we want to write to. Since our eeprom chip has 32,000 address locations we are using two bytes(16 bits) to store the address but we can only send one byte at a time so we have to split it up. The first send function takes the eeaddress and shifts the bits to the right by eight which moves the higher end of the 16 bit address down to the lower eight bits. Next we do a bitwise AND to get just the last eight bits. To illustrate this lets follow the steps below.

Lets say we want to write to address location 20,000 which is 0100 1110 0010 0000 in binary. We need to send the MSB(Most significant bits) first so we have to shift our address to the right eight bits.

0100 1110 0010 0000 (eeaddress)
After shifting 8 bits to the right we have
0100 1110

We now have the first half of the address, time to get the second half:

0100 1110 0010 0000 (eeaddress)
After we bitwise AND 0xFF with eeadddress we get
0010 0000

This means our 24LC256 chip gets the address 1001 1100 and then 0010

0000 which tells it to store the next byte in address location 20,000. Now that we've sent the address we send the data and then we end the process by calling the endTransmission function. The 24LC256 gets the data and writes the data to that address location. To finish up this function you'll notice I've included a delay of 5 milliseconds. This allows the chip time to complete the write operation, without this if you try to do sequential writes weird things might happen.

Now that you're data has been stored it's time to get it back, lets examine the readEEPROM function.

```
byte readEEPROM(int deviceaddress, unsigned int eeaddress )
{
  byte rdata = 0xFF;

  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // MSB
  Wire.send((int)(eeaddress & 0xFF)); // LSB
  Wire.endTransmission();

  Wire.requestFrom(deviceaddress,1);

  if (Wire.available()) rdata = Wire.receive();

  return rdata;
}
```

The readEEPROM accepts two arguments and returns on byte(the data). The arguments it accepts are the same first two arguments the write function, the device address and the address on the eeprom to read from. First we declare a variable to store the byte we're going to retrieve. Next we start off just like we did with the write function by starting the process with beginTransmission and then we send the address we want to access; this works exactly the same way as the write function. Continuing on we end the the transmission and we've now set the 24LC256 with the address we're interested in so now we just have to request and read the data. The next function requestFrom()

sends the command to that chip to start sending the data at the address we set above. The second argument is how many bytes(starting at this address) to send back; we're only requesting one. Finally we check to see if there is data available on the I²C bus and if there is we read it into the rdata variable. We return the byte of data and we're done!

That's all you really need to know in order to use and external I²C EEPROM chip with the Arduino. Take this setup and play around with it, see if you can figure out how to store more than one byte at a time or if you want a challenge try using more than one 24LC256 on the same I²C bus.

**See also**

Eeprom Page Write (Writing long strings to 24LC256)