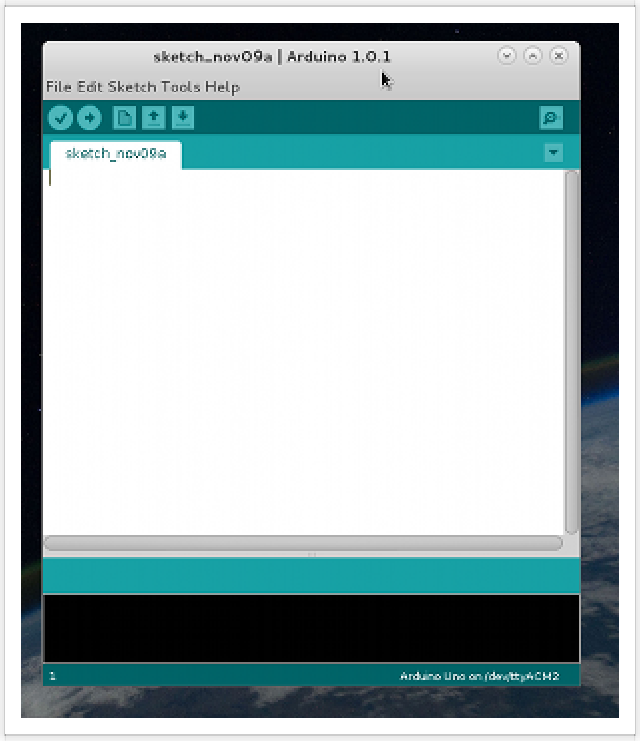
Using an Arduino as a PIC programmer

Ok, so I told you all last time that I was going to use one of these microcontrollers called a 'PIC' to control the motor speed on my robot, and that I was going to be cheap and use an Arduino to actually upload the program to the PIC. In this post I'm going to describe exactly how to do this, using some software written by a guy called Kirill Kulakov. You can read all about this on his blog, High Spark, but I'm going to try and give more of a step-by-step guide, and tell you what you need to change to use it with a different type of PIC (I'm using an 18F2420, Kirill originally had an 18F2550 in mind).

First, make sure you can compile some form of simple test code for the PIC. See my previous post for instructions. You should end up with a file called `LEDsOn.hex` - we'll make use of this later.

Next, you need to set up the Arduino environment. You might be able to get this from your package manager, or try here. Make sure you can compile and upload code to the Arduino (eg. this). There should be plenty of help available elsewhere on the web if you run into difficulty. You should be able to type `arduino &` into your command line and see something like this appear:


sketch_nov09a | Arduino 1.0.1
File Edit Sketch Tools Help
sketch_nov09a
1
Arduino Uno on /dev/ttyACM2

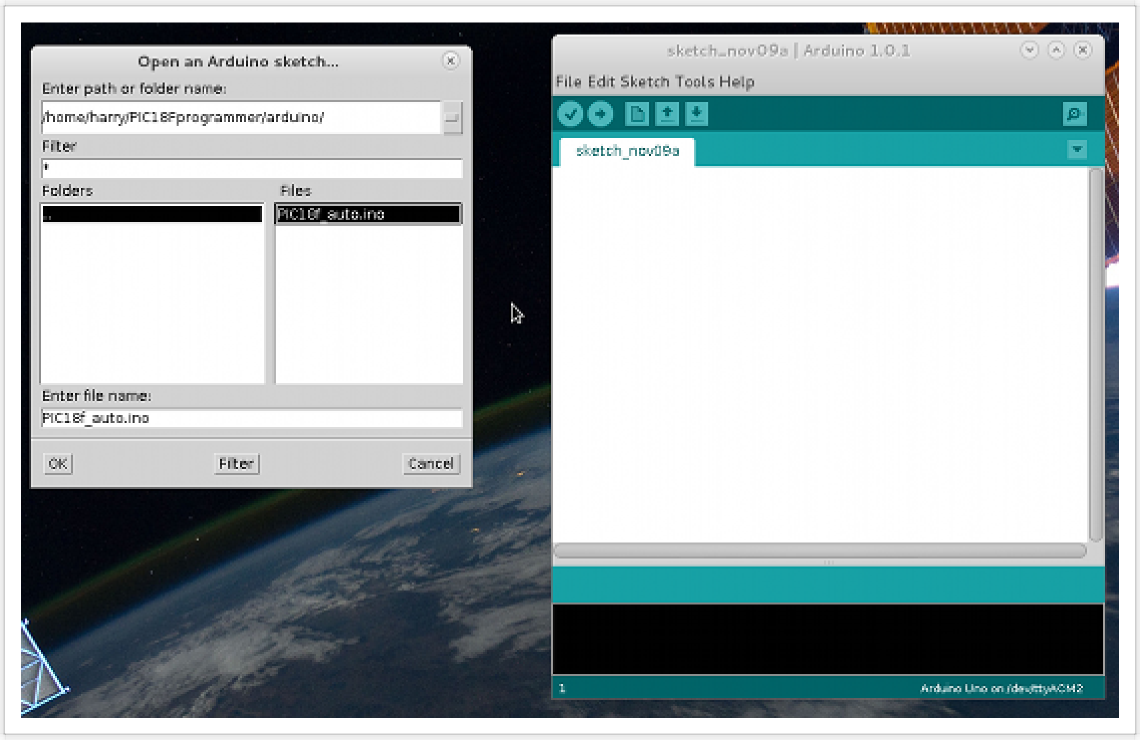
Connect you arduino and test that you can get the 'blink' sketch working. If this won't work, there's something wrong with your Arudino environment, or the arduino itself, or the connection to your PC.

Alright, let's assume you're able to upload programs to the arduino. We'll need two pieces of code to do our PIC programming:
1) Kirill's PIC programmer. This runs on the Arduino and pulls various pins high or low to write to the PIC. Think of it as translating bytes from our hex file into the PIC's language.
2) Jose Carlos Granja's Linux PIC programmer. This code is going to take the .hex file on the PC and read out bytes to the Arduino.

You can download both of these from a single github repository. Open a terminal, change to your home directory, and type

```
git clone https://bitbucket.org/JoseFuzzNo/arduino-as-pic18f-programmer-for-linux.git PIC18Fprogrammer
```

Open up the arduino environment, and open up the .ino file that lives in PIC18Fprogrammer/arduino/PIC18f_auto/



Open an Arduino sketch...

Enter path or folder name: /home/harry/PIC18Fprogrammer/arduino/

Filter: *

Files: PIC18f_auto.ino

Enter file name: PIC18f_auto.ino

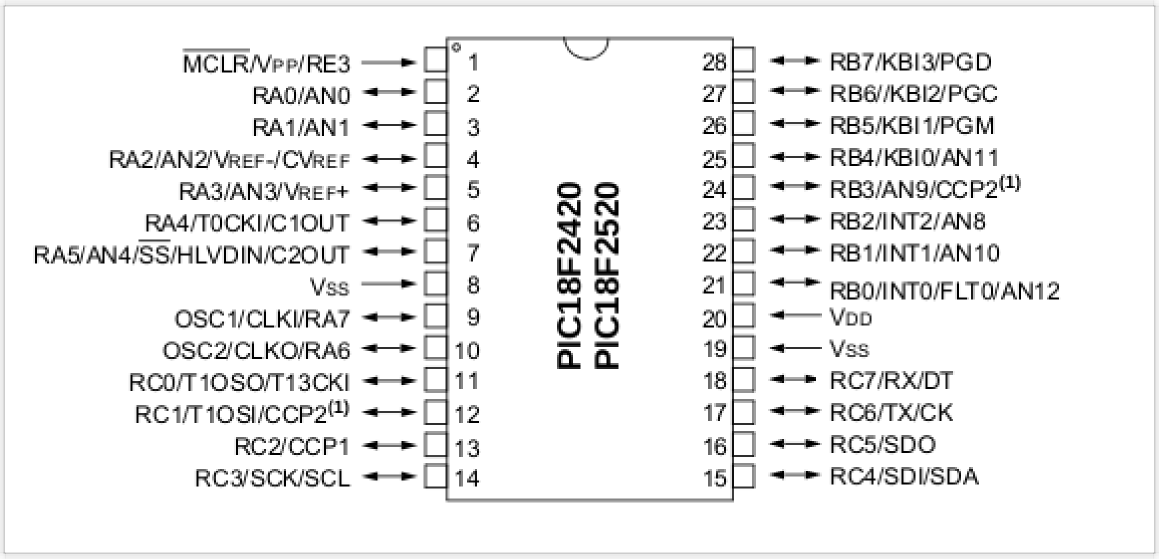
Folders
OK
Filter
Cancel
sketch_nov09a | Arduino 1.0.1
File Edit Sketch Tools Help
sketch_nov09a
1
Arduino Uno on /dev/ttyACM2

The environment will complain that it wants to move the file it its own directory. Click 'Ok'.

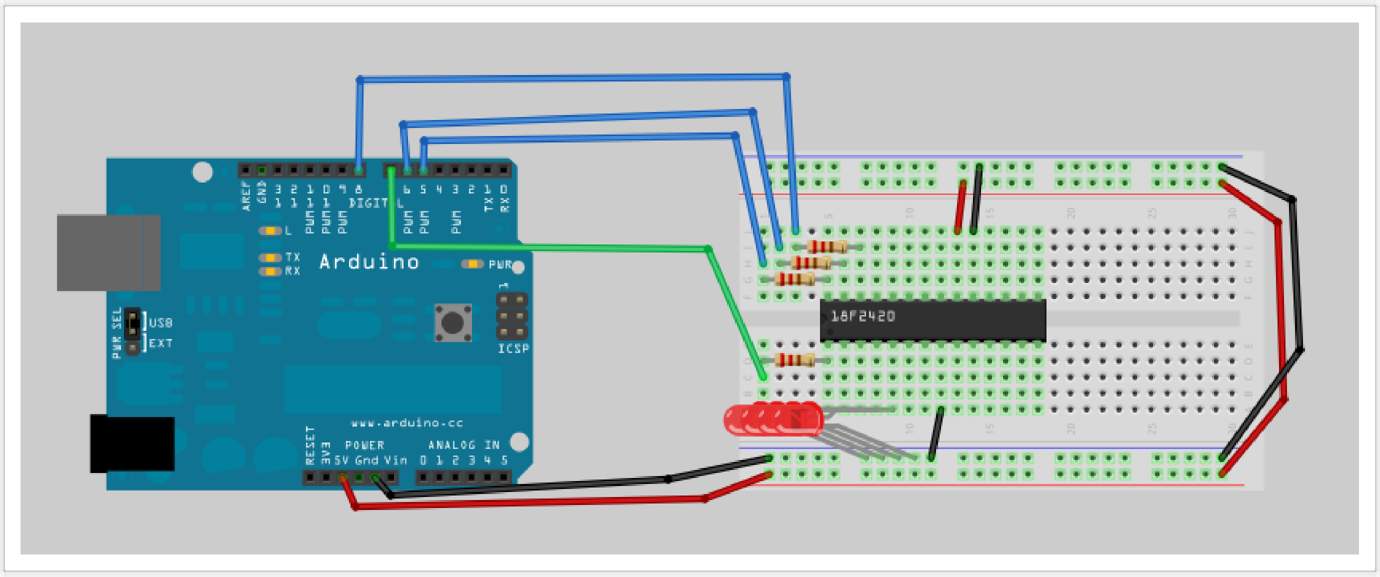
Ok, now let's wire up the arduino to program the PIC. You'll need the arduino itself, some kind of protoboard (I usethese), 4 resistors (at least 200 ohm each), a PIC 18F, and some wire.

We need to wire up the following pins on the PIC:
1) The VDD pin is the power input, this want to be connected to the 5V pin on the arduino.
2) The two VSS pins should be wired to ground.
3) The MCLR pin should be wired (via a resistor) to pin 7 of the arduino.
4) The PGC pin should be wired (via a resistor) to pin 6 of the arduino.
5) The PGD pin should be wired (via a resistor) to pin 5 of the arduino.
6) The PGM pin should be wired (via a resistor) to pin 8 of the arduino.
7) Connect an LED from each of RA0, RA1, RA2, RA3 to ground. The 'anode' (the longer leg of the LED) should be connected to the PIC, and the 'cathode' (the shorted leg) connected to ground. If you only have one LED, connect it to RA0 and ground. These are how we're going to see that our program has been uploaded successfully.

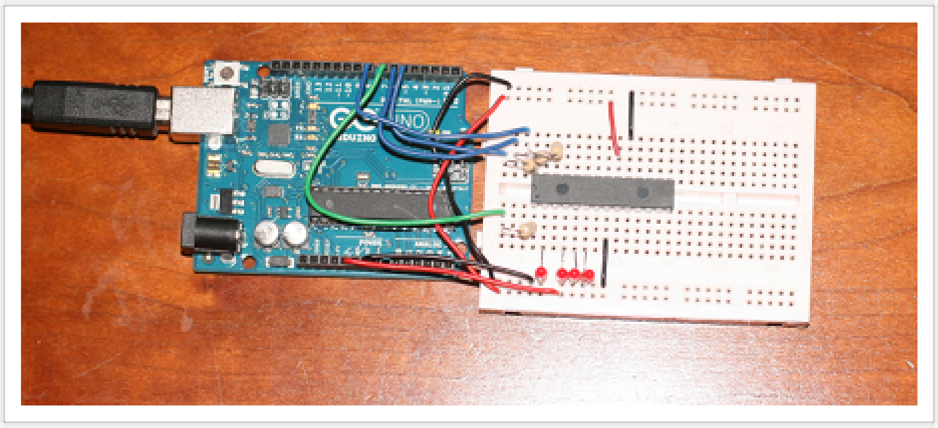
We get those last four by looking at the '#define' definitions at the top of the PIC18f_auto.ino file. How do I know what pins these correspond to on the PIC? I look in the reference manual for the PIC18F2420 and find the following diagram



Corresponding manuals exist for other models of PIC in the 18F family. Whilst I haven't tested them, I suspect this programming method with work fine with them too. Here's a diagram of how to wire this up for the 18F2420



and a photo (not quite identical wiring, but electrically the same, except that I've connected RA4 instead of RA1 with an LED - doesn't matter for this example)



Ok, so we have our hardware wired up. Attach the arduino to the PC via a usb cable, and hit the 'Upload' button (the arrow near the top left of the window) in the arduino environment. The arduino is now all ready to talk to the PIC. We need to build Jose's code so that the PC so can talk to the arudino. Change to the directory this code

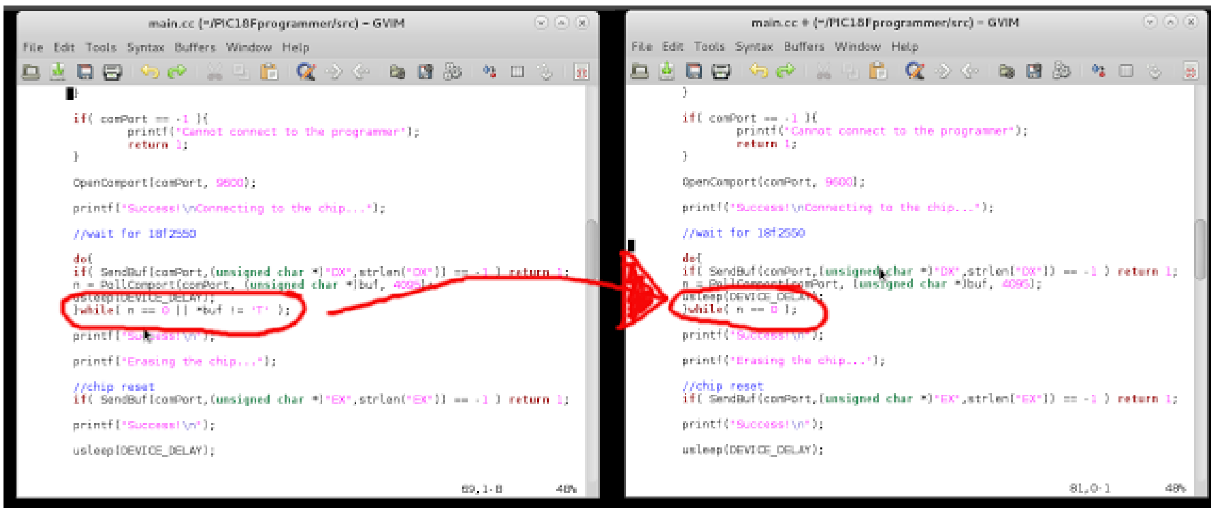
resides in
`cd ~/PIC18Fprogrammer/src/`
and open the file `main.cc` with your favourite text editor. We need to replace line 86, which originally reads

```
 }while( n == 0 || *buf != 'T' );
```

with

```
 }while( n == 0 );
```

Here it is in context:


main.cc (~/PIC18Fprogrammer/src) - GVIM
File Edit Tools Syntax Buffers Window Help
69,1-8 48%
main.cc + (~/PIC18Fprogrammer/src) - GVIM
File Edit Tools Syntax Buffers Window Help
81,0-1 48%

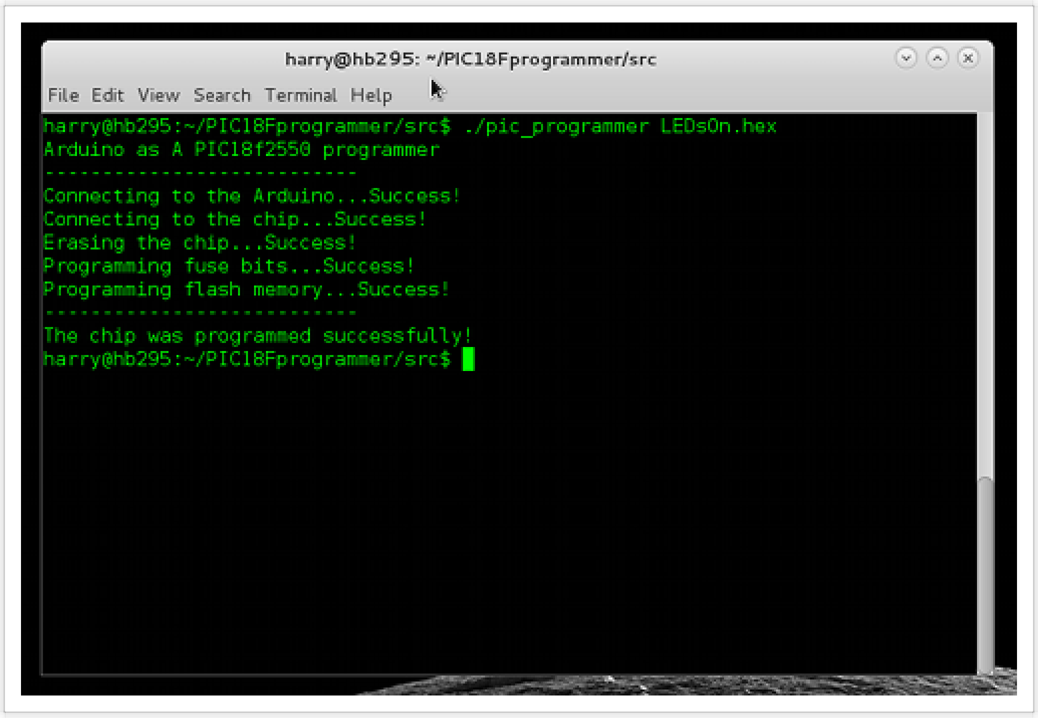
```
	}

	if( comPort == -1 ){
		printf("Cannot connect to the programmer");
		return 1;
	}

	OpenComport(comPort, 9600);

	printf("Success!\nConnecting to the chip...");

	//wait for 18f2550

	do{
	if( SendBuf(comPort,(unsigned char *)"DX",strlen("DX")) == -1 ) return 1;
	n = PollComport(comPort, (unsigned char *)buf, 4095);
	usleep(DEVICE_DELAY);
	}while( n == 0 || *buf != 'T' );

	printf("Success!\n");

	printf("Erasing the chip...");

	//chip reset
	if( SendBuf(comPort,(unsigned char *)"EX",strlen("EX")) == -1 ) return 1;

	printf("Success!\n");

	usleep(DEVICE_DELAY);
```

```
	}

	if( comPort == -1 ){
		printf("Cannot connect to the programmer");
		return 1;
	}

	OpenComport(comPort, 9600);

	printf("Success!\nConnecting to the chip...");

	//wait for 18f2550

	do{
	if( SendBuf(comPort,(unsigned char *)"DX",strlen("DX")) == -1 ) return 1;
	n = PollComport(comPort, (unsigned char *)buf, 4095);
	usleep(DEVICE_DELAY);
	}while( n == 0 );

	printf("Success!\n");

	printf("Erasing the chip...");

	//chip reset
	if( SendBuf(comPort,(unsigned char *)"EX",strlen("EX")) == -1 ) return 1;

	printf("Success!\n");

	usleep(DEVICE_DELAY);
```

Why are we doing this? Hose's original code basically asked the PIC "Are you an 18F2550?" and stopped if the PIC said "No". We want to program other types of PIC18F, so we're removing this test. Again, I've only tried programming an 18F2420, but I suspect this will work on many other models.

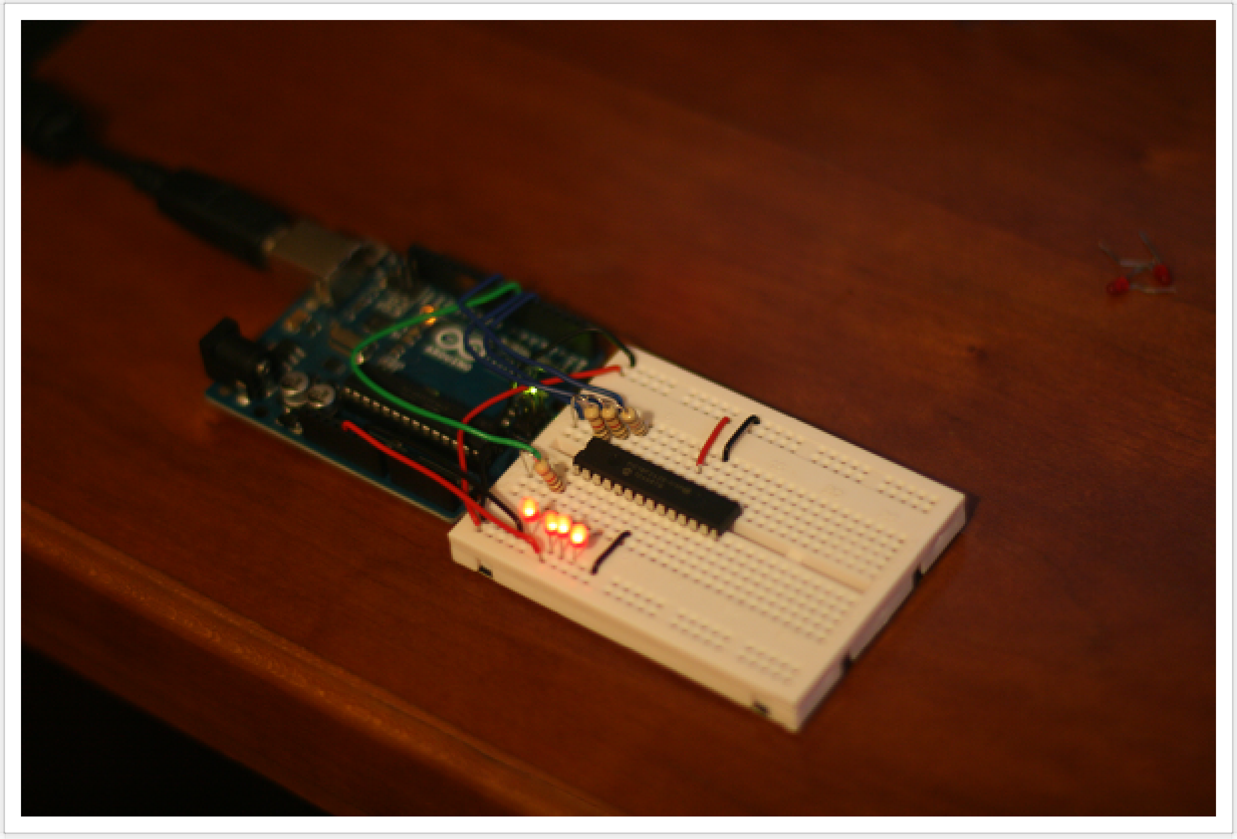
Compile the code by typing
`make`
into a terminal while in this directory.

Assuming the code makes without errors (don't worry about warnings), copy the LEDsOn.asm file that you made in this tutorial into the `~/PIC18Fprogrammer/src/` directory. Make sure the arduino environment window is still open. In your terminal window type
`./pic_programmer LEDsOn.hex`
If all went well you should see something like this:


harry@hb295: ~/PIC18Fprogrammer/src
File Edit View Search Terminal Help

```
harry@hb295:~/PIC18Fprogrammer/src$ ./pic_programmer LEDsOn.hex
Arduino as A PIC18f2550 programmer
---------------------------
Connecting to the Arduino...Success!
Connecting to the chip...Success!
Erasing the chip...Success!
Programming fuse bits...Success!
Programming flash memory...Success!
---------------------------
The chip was programmed successfully!
harry@hb295:~/PIC18Fprogrammer/src$
```

But never mind that! Those four LEDs should now have switched on!



Success!

I'll put up a new post in the future about how to make the PIC do something more interesting (and more robot-oriented ;-)). I'll also explain how to compile programs written in C, which is much nicer to work with than assembler.

Finally, if you want quick access to a copy of the code with the necessary modifications and the LEDsOn.hex programme already included, clone the copy from my git repository:

```
git
clone https://github.com/harrybraviner/PIC18Fprogrammer_for_blog.git PIC18Fprogrammer
```