

Getting Started with the LPC810

Created by Kevin Townsend



https://learn.adafruit.com/getting-started-with-the-lpc810

Last updated on 2022-12-01 01:57:58 PM EST

Table of Contents

Introduction	3
Setting up an ARM Toolchain	4
Downloading the LPCXpresso IDE Installing LPCXpresso	
Importing a Project Into LPCXpresso	7
Blinky!	10
Building Firmware in LPCXpresso	
Programming the LPC810 with Flash Magic	13
Programming the LPC810	
Hooking Everything Up	
Using Flash Magic	
• Get Your .HEX File Ready	
Configuring Flash Magic for the LPC810	
Checking your Connection	
Programming the Device	
Testing Your Firmware	
Sample Hex Files	
OK But why the LPC810 at Adafruit?	20
• 'Small is Beautiful'	

Introduction



The LPC810 is a new microcontroller from NXP, a super fast little controller has only 8 pins but packs a lot of punch with a high speed processor and 32 bit instructions. This learning guide will show you everything you need to know to get started with the ARM Cortex M0+ based LPC810 microcontroller. It will cover:

- Setting up a cross-compiling toolchain for ARM
- Creating and compiling your first blinky program
- Programming the LPC810 using free and open source tools

This tutorial is designed to be followed using the <u>Adafruit LPC810 Mini Starter Pack</u> (ht tp://adafru.it/1336)



Setting up an ARM Toolchain

While there are literally dozens of choices out there for compilers and IDEs for ARM, we're partial to GCC, which tends to have particularly good support for ARM chips in general.

While most seasoned veterans probably prefer a good old makefile, if you're new to ARM you'll appreciate the hand-holding an IDE can provide. We'll be using NXP's free LPCXpresso IDE in this tutorial, which is based on Eclipse and GCC, and works with all of their recent chips.

If you don't wish to use LPCXpresso, most of these steps also apply to a vanilla Eclipse + GCC installation, but you will have to do a bit more work yourself since LPCXpresso allows you to cut a lot of corners, generating the makefile and linker script on the go, etc. If there's enough interest, we'll also put together a tutorial on creating a makefile and building from the command line for this chip using only vanilla GCC.

Downloading the LPCXpresso IDE

In order to download NXP's free LPCXpresso IDE, you do need to create an account on Code Red's website (who are now owned by NXP), at https://www.lpcware.com/lpcxpresso/download ()

While registering sucks, LPCXpresso is still the easiest free tool using an open source toolchain, and you can easily break out of it once you're a bit more comfortable with ARM and GCC, creating your own makefile and linker script, and LPCXpresso is just a bit of icing on top of the Eclipse + GCC cake.

An advantage of LPCXpresso is that it includes a relatively easy to use installer for Linux, Windows and Mac OS X, so you're free to use the toolchain and IDE on any major platform.

After registering, you should see a download list similar to the following:

LPCXpresso 5 for Windows			
Goto Downloads			
Welcome to the LPCXpresso for Windows pages. Note: If you are upgrading from a previous installation, we recommend that you install into a new directory. Your existing activation code will be used.			
FTP Downloads			
LPCXpresso v5.2.4 Build 2122			
LPCXpresso v5.2.2 Build 2108			
LPCXpresso v5.1.2 Build 2065			
Still having problems? Try the LPCXpresso support forum			
Top of page			
Attachment	Size		
LPCXpresso_5.2.4_2122.exe	287.41 MB		
LPCXpresso_5.2.2_2108.exe 283.67 MB			
LPCXpresso_5.1.2_2065.exe	266.69 MB		

These are the latest downloads for Windows, but LPCXpresso is also available for LInux and MAC OS X.

Installing LPCXpresso

The installation process is relatively straight forward, and it should be fairly foolproof. Starting with the welcome screen, just run through the installer, and LPCXpresso will automatically setup all the tools, including the cross-compiling toolchain, Eclipse, etc.



At the end of the installation process you can optionally look at some documentation on how to use LPCXpresso by selecting or deselecting the checkboxes, but at this point you've installed everything you need to start writing programs for the LPC810 (or any other LPC chip from NXP)!



Next you can start the LPCXpresso IDE (you may need to activate it if this is the first time using it), and you'll get a screen similar to the following:

Develop - Welcome =>percub/Capers			= Ø X
File Edit Navigate Search Project Run Window He			
😅 = 🗟 🛆 0 = % = [@ 🕸 🖌 9 👜 = .	C C C C C C C C C C C C C C C C C C C	9 • 0 • 9 • 1 6 (\$100 H # 17 3) 3) 15 \$2 (\$ \$2 \$4 • 18 B [\$ • 8 • 9 • 9 • 9 • 1 5]	Quick Access
Co Project Ex 12 7, Periphera 122 Registers	@ Welcome 11		
8 🕲 🎽	🖕 🖒 🔳 🤌 file///C/mg/LPC/presss 5.2.4.2122/pc/presss/pages/register	ed Jon	- F 🖬
		PERFESSO powerd by Scode_red	
		LPCXpresso5 is fully activated	
		Welcame in LPCRamme 5. The software is now fully activated, and can be used for production. LPCRamme can be used to generate and download applications containing up to 120920 of code into an LPC target.	
		Product Documentation	
		Full product documentation is provided within your tools suite installation via the IDE menu: a Itelp > Help Contents	
		NXP Getting Started Guide	
		The KEP LPCEpressa Getting Started Guide provides step-by-step instructions to build your first LPCEpressa project.	
		Red Trace Guide	
		The Trace Guide provides information on using instruction Trace within LPCXpresso. LPCXpresso Renzoances	
		CPC/gresso resources Ones the LPC/gresso home page for new LPC/gresso software releases, more example projects, and announcements.	
		 Visit wave.code-red inch.com/progressio-home 	
		LPC/gresso Forum	
∪ Q_ = 22 ++ Y = 4 ₀ 0− <u>12</u> 0− 4 <u>7</u> 0− = □		LOCAPERSO FORM The LFCQUERSO Forms formatily is exclusively built for LFCQuerses users and is an invaluable resource for getting help and advice on use of the LFCQuerses tomband builty. So, get consorted and gain the LFCQuerses community!	
45 Starthere 8		* To view the forum, and to register so that you can post, visit www.code.red.tech.com/propresso-forum	
2 New project		General Information	
Import project(s)		* For information on KOP Microcontrollers vial www.nxp.com/products/microcontrollers	
🔛 Build all projects []		» For information on upgrades etc. visit the Code Red Technologies LPCEpresso website » Visit the Endedded Artists LPCEpresso website	
4 0-011 D		* You be calender where the appears where	
/ Cess*[]			
St Delrog * []			
Quick Settings *			
🗧 Project and File wigards 🛛 🗧	Console 22 Problems Memory Instruction Trace Red Trace Ne consoles to display at this time.	Protes	d 🛛 - 😆 - " o
K Import and Export 8			
🐔 Build and Settings 🛛 🗧			
RC Debug and Run X			
W Laters W			
workspace			

Importing a Project Into LPCXpresso

The easiest way to get started with the LPC810 is to use an existing project, where everything has already been setup for you in Eclipse. We'll use the code base available available here for this tutorial: https://github.com/microbuilder/LPC810_CodeBase ()

The first thing to do is download a .zip file containing all of the files in the project. You can do this by going to the github project page () and clicking the 'ZIP' button shown below:

Clone in Windows			
branch: master 💌	Files	Commits	
LPC810_CodeBase / 관			

Once you have this .zip file available somewhere on your computer, open up the LPCXpresso IDE, and click the "Import Project(s)" button in the bottom left-hand side:

🕛 Quickstart Panel 🔀	
📳 Start here	*
📸 New project	
P Import project(s)	
Build all projects [Release]	
K Build 'LPC810_CodeBase' [Release]	
🧹 Clean 'LPC810_CodeBase' [Release]	
🕸 Debug 'LPC810_CodeBase' [Release]	
🖄 Quick Settings 🔻	
Project and File wizards	×
📳 Import and Export	×
📳 Build and Settings	×
📒 Debug and Run	×
📒 Extras	×

This will bring up a dialogue box, and you need to click the 'Browse...' button under 'Project Archive (zip)', highlighted below:

🗿 Import project(s)	3
Import project(s)	
Select the examples archive file to import.	
Projects are contained within archives (.zip) or are unpacked within a directory. Select your project archive or root directory and press <next>. On the next page, select those projects you wish to import, and press <finish>.</finish></next>	
Project archive (zip)	
Archive Browse	
Project directory (unpacked) Root directory Browse	
Browse for more examples	
Press "Browse for more examples" to view the latest examples and download to your local drive. Then use press "Browse" (above) to import into your workspace.	
(?) < Back Next > Finish Cancel	

Select your .zip file, and then click the Next > button, which will bring up the following dialogue:

Import project(s)		~	
Import project(s) Select a directory to search for existing Eclipse projects.			
Projects:			
LPC810_CodeBase (LPC810_CodeBase-master)			Select All Deselect All Refresh
☑ Copy projects into workspace	-		
?	< Back Next >	Finish	Cancel

Just click Finish, and your project will be imported into your workspace. You should see an entry in the project explorer on the left hand side named 'LPC810_CodeBase', shown below:



That's everything we need to know to start writing our own program with this chip!

Blinky!

Now that we have our project imported, we can get start creating our own projects.

By default, the LPC810_CodeBase you downloaded is setup to run a blinky example on pin 0.2, and it spits out 'Hello, LPC810!' on the default UART pins, which we can see in the 'while(1)' super-loop in main.c, shown below:

```
while(1)
{
 #if !defined(USE_SWD)
    /* Turn LED Off by setting the GPIO pin high */
   LPC_GPI0_PORT->SET0 = 1 <&lt; LED_LOCATION;
   mrtDelay(500);
    /* Turn LED On by setting the GPIO pin low */
   LPC_GPI0_PORT->CLR0 = 1 <&lt; LED_LOCATION;
   mrtDelay(500);
 #else
   /* Just insert a 1 second delay */
   mrtDelay(1000);
 #endif
  /* Send some text (printf is redirected to UARTO) */
  printf("Hello, LPC810!\n");
}
```

"LPC_GPIO_PORT->CLR0 = 1 << LED_LOCATION" will clear whatever pin is at the 'LED_LOCATION' bit on GPIO bank 0. 'LPC_GPIO_PORT->SETO' does the opposite. This is how you turn the LED on or off, but for those changes to be visible, we need to insert a delay between state changes, which is accomplished with the mrtDelay(500) lines, which causes a 500ms delay.

Blinky is following by our printf() statement, and by default all printf output is redirect to UARTO on the LPC810.

Before worrying about writing and programs, though, lets build the firmware as-is!

Building Firmware in LPCXpresso

To build a project in LPCXpresso (or Eclipse), you need to right-click on the current project in the project explorer (located on the left-hand side of the IDE by default), and select the 'Build Project' menu item.



If everything goes well (and there are no errors in our code), you should get something like this in the output window at the bottom of LPCXpresso to indicate that the build process is complete:

📃 Console 🕱 🖹 Problems 📋 Memory 🌾 Instruction Trace 📲 Red Trace Preview
CDT Build Console [LPC810_CodeBase]
makeno-print-directory post-build
Performing post-build steps
arm-none-eabi-size "LPC810_CodeBase.axf"; arm-none-eabi-objcopy -0 binary "LPC81
"LPC810 CodeBase.bin";
text data bss dec hex filename
2228 4 4 2236 8bc LPC810 CodeBase.axf
Created checksum 0xeffff949 at offset 0x1c in file LPC810_CodeBase.bin
00:27:37 Build Finished (took 25.752ms)
,

We now have our 'blinky' firmware (located in the same folder as our project files), and can get it onto our LPC810 using the UART bootloader built into the chips ...

Programming the LPC810 with Flash Magic

OK now is the time to pull out your <u>Adafruit LPC810 Mini Starter Pack</u> (http://adafru.it/ 1336)



Programming your LPC810 is relatively foolproof, and these chips are extremely hard to brick since they have a ROM-based bootloader and 'safety mode' that can always be executed before any bad firmware we might have programmed into the chip. We'll use this feature to program the MCUs with the firmware we built in the previous step.

But first ... a bit of wiring is in order! You'll need to hookup a UART to USB adapter, a 3V3 voltage regulator, a couple caps, and a few wires to get everything wired up safely with the LPC810. Just follow the wiring diagram below and you should be good.

The UART cable colors on the right (four wires) match the colors of Adafruit's USB to TTL Serial Cable (http://adafru.it/954).



Programming the LPC810

There are two main ways to take advantage of ISP mode to program your MCU:

- Use Flash Magic (), a free GUI-based tool for Windows
- Use lpc2isp (), an open source command-line tool that can be adapted to work on most operating systems, though you may need to build it yourself and add your MCU ID.

This tutorial will focus on Flash Magic as a starting point since it's the easiest tool to use, and since many Linux users should be able to figure out lpc2isp without too much effort.

Hooking Everything Up

This tutorial will assume that you are using the PL2303-based USB to TTL Serial Cable (http://adafru.it/954) included with the LPC810 Starter Kit.

- 1. Insert the LPC810 into your breadboard as shown, along with the 3.3V voltage regulator, and two decoupling capacitors.
- 2. Connect the white, green, red and black wires on the right-hand side of the diagram to the same colored cables on your USB to TTL serial adapter.
- 3. Connect the ISP pin to GND (the cable shown in gray above), which will cause the chip to enter ISP mode when it comes out of reset
- 4. Connect the other red and black cables on the breadboard which control the power supply to the MCU.

Note: Be sure to connect the ISP pin to GND before powering the board (with the red and black cables), since the LPC810 checks this pin as soon as it has power to device if it should enter ISP mode or not. When the ISP is at GND on boot, the

chip enters ISP mode, when ISP is floating or logic high, th chip will boot normally, skipping ISP mode.

Using Flash Magic

If you haven't already done so, download and install the latest version of the free $\underline{\mathsf{Flas}}$ h Magic () tool.

The tool is relatively straight forward once you have used it once or twice, but this guide will show you everything you need to know to program the LPC810 using the Flash Magic GUI.

Get Your .HEX File Ready

Flash magic uses Intel Hex files to program the MCUs.

The LPC810 CodeBase () will produce a compatible .hex file from your own code, which we'll learn about later, but you can also find some sample .hex files at the bottom of this tutorial for convenience sake.

Configuring Flash Magic for the LPC810

Before you can program your LPC810, you need to supply some basic information to Flash Magic:

- Set the device to 'LPC810M021FN8'
- Set the COM port to whatever COM port your USB to TTL cable is using (you can find this in the Device Manager on Windows, for example)
- Set the Baud Rate to '115200'
- Set the Oscillator to '12'
- Check the 'Erase blocks used by Hex File' checkbox

Once these settings are entered (see the image below for a reference), you simply need to point to your .hex file by clicking on the 'Browse...' button

🌧 Flash Magic - NON PRODUCTION USE ONLY			
File ISP Options Tools Help			
🖻 🖬 🔍 🗿 🐗 🗸 👅 🔈 💙 國 🌚 :	8		
Step 1 - Communications	Step 2 - Erase		
Select LPC810M021FN8	Erase block 0 (0x000000-0x0003FF) Erase block 1 (0x000400-0x0007FF)		
Flash Bank:	Erase block 2 (0x000800-0x000BFF)		
COM Port. COM 7 -	Erase block 3 (0x000C00-0x000FFF)		
Baud Rate: 115200 🗸	Erase all Elash+Code Bd Prot		
Interface: None (ISP) 🔹	Erase blocks used by Hex File		
Oscillator (MHz): 12	_ ,		
Step 3 - Hex File			
Hex File:	Browse		
Modified:Unknown	more info		
Step 4 - Options	Step 5 - Start!		
 Verify after programming Fill unused Flash 	Start		
Gen block checksums			
Execute			
🗌 Activate Flash Bank			
On-Line training classes for microcontrollers and embedded networking and Internetworking www.esacademy.com/faq/classes			
	0		

Checking your Connection

You can make sure that all of your settings are correct, and that everything is connected properly and that the chip is actually in ISP mode by selecting 'ISP > Read Device Signature ...' from the top menu bar:



If everything is OK, you should see something similar to this:

🙀 Flash I File ISP	Magic - NON PRODUCTIO Options Tools Help					23
	Q 🗿 🏈 🗸 🎩 >		2			
Step 1 -	Communications		Step 2 - Eras	e.		
Selec	t LPC810M021FN8			0 (0x000000-0		
Flash B	ank:	-		1 (0×000400-0 2 (0×000800-0		
СОМ	Device Signature				×	
Baud	Manufacturer ID: 0x					
Inter	Device ID 1: 0x				_	
Oscill	Device ID 2: 0x					
_	Device ID: 0x	00008100				
Step 3						
Hex F	Bootloader Ver:	13.2			se.	
- 1	Serial Number:	84111398 292	5627493 13544	64379 411041;	8566	
Step 4						
🗖 Verif					Close	
🔲 Fill u	lock checksums	_	_	_		
Execu						
Activa	ate Flash Bank					
Rotating, fully customizable, remotely updated Internet links. Embed them in your application! www.embeddedhints.com						
				0		

If you get an error message, check your COM port and the connections to your USB to TTL cable and ensure that the ISP pin is pulled low to GND before powering or resetting your board.

Programming the Device

After selecting your .hex file in the 'Hex File' textbox, you simply need to click the Star t button, and Flash Magic should start programming your device.

You may wish to check the 'Verify After Programming' checkbox before doing this, but it isn't strictly necessary:

🌧 Flash Magic - NON PRODUCTION USE ONLY				
File ISP Options Tools Help				
🖻 🖬 🍳 🎯 🗳 🖌 🎩 🔈 😎 🚳 🎯 🎗	3			
Step 1 - Communications	Step 2 - Erase			
Select LPC810M021FN8	Erase block 0 (0x000000-0x0003FF) Erase block 1 (0x000400-0x0007FF)			
Flash Bank:	Erase block 2 (0x000800-0x000BFF)			
COM Port: COM 7 -	Erase block 3 (0x000C00-0x000FFF)			
Baud Rate: 115200	Erase all Flash+Code Bd Prot			
Interface: None (ISP)	Erase blocks used by Hex File			
Oscillator (MHz): 12				
Step 3 - Hex File				
Hex File: D:\Shared\Dropbox\microBuilder\Co	de\LPC812\LPC810_CodeBa Browse			
Modified:Friday, April 5, 2013, 14:05:13	more info			
Step 4 - Options	Step 5 - Start!			
Verify after programming Start Fill unused Flash Start Gen block checksums Execute Activate Flash Bank Start				
Technical on-line articles about 8051 and XA programming				
www.esacademy.com/faq/docs				
	1			

Testing Your Firmware

At this point, your device should be programmed. To test your firmware perform the following steps:

- Disconnect the ISP wire on your breadboard (the gray wire in the image above)
- Reset the device ... for example, disconnect then reconnect the red power cable on your USB to TTL adapter
- Once the board is powered up and if the ISP pin is not pulled low the chip will execute code normally

Sample Hex Files

For convenience sake, you can find some pre-compiled sample .hex files for the LPC810 in the '/tools' folder of the LPC810 code base (), available on github:

LPC810_Blinky_P0_2.hex - This hex file will cause an LED on P0.2 to blink at a regular rate. To use this firmware, connect the anode (the larger pin on your LED) to P0.2 on the LPC810, and the cathode (the shorter pin on the LED) to a 1K resistor and to GND, as seen in the following diagram:



OK ... But why the LPC810 at Adafruit?

I'll be honest ... I just thought this was an incredibly interesting chip. I've been using NXP's LPC series of MCUs since, oh, forever ... but this particular chip in DIP8 really jumped out at me since it's so different than what people usually think of when they hear 'ARM'.

The DIP8 LPC810 is still somewhat of a challenge to use precisely because it's so small (by ARM standards, anyway): 4KB flash and 1KB SRAM. That doesn't go far, and in reality you only have about 3KB flash to play with once you add in your startup code and get everything setup.

But that's actually part of what I found so fun about this chip! It's a genuine challenge but a fun one to do something creative and interesting with so much performance in such a small space!

'Small is Beautiful'

Warning: Shameless, unadulterated, highly-personal rant!

Really ... <u>small is where it's at</u> () if you care about writing clean, efficient code and really learning how to do something properly and understanding what you're doing!

I always start my new projects with the smallest chip I can, rather than the biggest one available. Why? Because it forces me to keep size in mind, to try to write better, more efficient code, and to get the most out of the limited resources I have. Why do I always start with <u>ARM chips with 32KB flash and 8-12KB SRAM</u> () when I could get something with 512KB flash and 100KB+ SRAM for not much more? Because if I started with those big chips I'd write even sloppier code that took more space than I needed, and I'd be stuck with bigger and more expensive chips and even worse code forever!

One of the biggest difficulties in deeply embedded systems is keeping things lean, without compromising on stability and reliability. That means error checking, data

validation, etc., which takes space and time to implement, but you need to keep space in check, so it's an endless series of trade offs, decisions and optimisations.

Deeply embedded development has a lot more in common with mechanical watchmaking than it does with traditional SW development: you're always trying to fit an impossible seeming number of gears and components into a ridiculously small package, and everything has to fit in in just the right manner to work at all. To throw some oil on the fire, it also needs to hold up to all manner of abuse and mistreatment that will get thrown at it, and keep smiling back at you day after day!

Something embedded SW engineers understand better than almost anyone else -except maybe those crazy mechanical watch makers -- is the huge satisfaction in writing clean, concise, efficient code that solves real problems in concrete, elegant, reliable ways. Writing bloated code is easy ... writing elegant code and creating tiny solutions for big ambitions is harder, but also infinitely more satisfying!

Why the LPC810 with it's tiny package, and limited resources? Because it's a fun challenge and a really interesting way to learn ARM!

I really wanted to get this chip in people's hand just to see what people can do with 4KB flash and 1KB SRAM, and a reasonably limited set of peripherals. It's the kind of fun challenge that I enjoy working on, and I'm sure I'm not alone in that!