



## The LPC824 Microcontroller System

Copyright © 2017-2022, Jesus Calvino-Fraga. Not to be copied, used, or revised without explicit written permission from the copyright owner.

### Introduction

This document introduces a minimal microcontroller system using NXP's LPC824 microcontroller. The LPC824 IC is an ARM® Cortex®-M0+ microcontroller, featuring 32-Bit, 30MHz, 32KB (32K x 8) FLASH, in a 20-TSSOP.

### Recommended documentation

[LPC82x User manual - UM10800](http://www.nxp.com/documents/user_manual/UM10800.pdf): [http://www.nxp.com/documents/user\\_manual/UM10800.pdf](http://www.nxp.com/documents/user_manual/UM10800.pdf)

[LPC82x Product data sheet](http://www.nxp.com/documents/data_sheet/LPC82X.pdf) "32-bit ARM Cortex-M0+ microcontroller; up to 32 kB flash and 8 kB SRAM; 12-bit ADC; comparator": [http://www.nxp.com/documents/data\\_sheet/LPC82X.pdf](http://www.nxp.com/documents/data_sheet/LPC82X.pdf)

### Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the LPC824 microcontroller system used in ELEC291/ELEC292. It can be assembled using a bread board. Table 1 below lists the components needed to assemble the circuit.

Quantity	Digi-Key Part #	Description
2	BC1148CT-ND	0.1uF ceramic capacitors
2	BC1157CT-ND	1uF ceramic capacitors
1	1.0QBK-ND	1kΩ resistor
1	330QBK-ND	330Ω resistor
1	67-1102-ND	LED 5MM RED
1	67-1108-ND	LED 5MM GREEN
1	MCP1700-3302E/TO-ND	MCP17003302E 3.3 Voltage Regulator
1	N/A	BO230XS USB adapter
1	568-11619-1-ND	LPC824M201JDH20J
1	1528-1066-ND	20-TSSOP to DIP20 adapter board
2	A26509-10-ND	10-pin header connector
2	P8070SCT-ND	Push button switch

**Table 1. Parts required to assemble the LPC824 microcontroller system.**

Before assembling the circuit in the breadboard, the LPC824 microcontroller has to be soldered to the 20-TSSOP to DIP20 adapter board. This task can be accomplished using a solder iron as described in this video:

<https://www.youtube.com/watch?v=8yyUIABj29o>

Once the microcontroller is soldered to the adapter board, solder the two 10-pin header connectors. The assembled bread boarded circuit should look similar to the one in figure 1. Notice that the LPC824 works with a VDD voltage between 1.8V and 3.6V. For this reason a voltage regulator is required to convert the USB adapter 5V to 3.3V.



## Setting up the Development Environment on Windows

To establish a workflow for the LPC824 we need to install the following three packages:

### 1. CrossIDE V2.25 (or newer) & GNU Make V4.2 (or newer)

Download CrossIDE from: [http://ece.ubc.ca/~jesusc/crosside\\_setup.exe](http://ece.ubc.ca/~jesusc/crosside_setup.exe) and install it. Included in the installation folder of CrossIDE is GNU Make V4.2 (make.exe, make.pdf). GNU Make should be available in one of the folders of the PATH environment variable in order for the workflow described below to operate properly. For example, suppose that CrossIDE was installed in the folder “C:\crosside”; then the folder “C:\crosside” should be added at the end of the environment variable “PATH” as described [here](#)<sup>1</sup>.

Some of the Makefiles used in the examples below may use a “wait” program developed by the author. This program (and its source code) can be downloaded from the course web page and must be copied into the CrossIDE folder or any other folder available in the environment variable “PATH”.

### 2. GNU ARM Embedded Toolchain.

Download and install the GNU ARM Embedded Toolchain from:

<https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads>.

The “bin” folder of the GNU ARM Embedded Toolchain must be added to the environment variable “PATH” in order for the workflow described below to operate properly. For example, if the toolchain is installed in the folder “C:\Programs\GNU Tools ARM Embedded”, then the folder “C:\Programs\GNU Tools ARM Embedded\5.4 2016q2\bin” must be added at the end of the environment variable “PATH” as described [here](#)<sup>1</sup>.

Notice that the folder “5.4 2016q2” was the folder available at the time of writing this document. For newer versions of the GNU ARM Embedded Toolchain this folder name will change to reflect the installed version.

### 3. LPC Flash Loader.

Available in the web page for the course is the program “lpc21isp.zip”<sup>2</sup>. Download and decompress the archive file “lpc21isp.zip” somewhere in your hard drive.

The “bin” folder of the LPC Flash loader must be added to the environment variable “PATH” in order for the workflow described below to operate properly. For example, if the LPC824 Flash loader is installed in the folder “C:\CrossIDE\lpc21isp”, then the folder “C:\CrossIDE\lpc21isp\bin” must be added at the end of the environment variable “PATH” as described [here](#)<sup>1</sup>.

## Workflow.

The workflow for the LPC824 microcontroller includes the following steps.

---

<sup>1</sup> <http://www.computerhope.com/issues/ch000549.htm>.

<sup>2</sup> Lpc21isp.zip is derived from lpc21isp available from <https://sourceforge.net/projects/lpc21isp/>. The program was modified in order to integrate it with both CrossIDE and GNU Make.

## 1. Creation and Maintenance of Makefiles.

CrossIDE version 2.24 or newer supports project management using simple Makefiles by means of GNU Make version 4.2 or newer. A CrossIDE project Makefile allows for easy compilation and linking of multiple source files, execution of external commands, source code management, and access to microcontroller flash programming. The typical Makefile is a text file, editable with the CrossIDE editor or any other editor, and looks like this:

```
# Since we are compiling in windows, select 'cmd' as the default shell. This
# is important because make will search the path for a linux/unix like shell
# and if it finds it will use it instead. This is the case when cygwin is
# installed. That results in commands like 'del' and echo that don't work.
SHELL=cmd
# Specify the compiler to use
CC=arm-none-eabi-gcc
# Specify the assembler to use
AS=arm-none-eabi-as
# Specify the linker to use
LD=arm-none-eabi-ld
# Flags for C compilation
CCFLAGS=-mcpu=cortex-m0 -mthumb -g
# Flags for linking
LDFLAGS=-T lpc824_linker_script.ld --cref -nostartfiles
# List the object files involved in this project
OBJS=init.o main.o

# The default 'target' (output) is main.elf and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create main.elf.
# The linked file is converted to hex using program objcopy.
main.elf: $(OBJS)
    $(LD) $(OBJS) $(LDFLAGS) -Map main.map -o main.elf
    arm-none-eabi-objcopy -O ihex main.elf main.hex

# The object file main.o depends on main.c. main.c is compiled
# to create main.o.
main.o: main.c
    $(CC) -c $(CCFLAGS) main.c -o main.o

# The object file init.o depends on init.c. init.c is
# compiled to create init.o
init.o: init.c
    $(CC) -c $(CCFLAGS) init.c -o init.o

# Target 'clean' is used to remove all object files and executables
# associated with this project
clean:
    del $(OBJS)
    del main.elf

# Target 'FlashLoad' is used to load the hex file to the microcontroller
# using the flash loader.
FlashLoad:
    lpcprog.exe main.hex -ft232 115200 12000000

# Phony targets can be added to show useful files in the file list of
# CrossIDE or execute arbitrary programs:
dummy: lpc824_linker_script.ld main.hex
    @echo :-

explorer:
    @explorer .
```

The preferred extension used by CrossIDE Makefiles is “.mk”. For example, the file above is named “blinky.mk”.

Makefiles are an industry standard. Information about using and maintaining Makefiles is widely available on the internet. For example, these links show how to create and use simple Makefiles.

<https://www.gnu.org/software/make/manual/make.html>

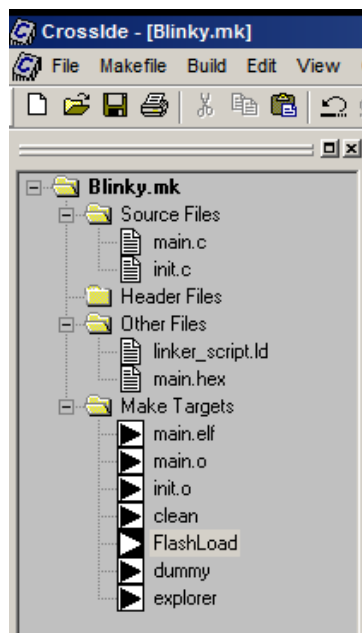
<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>

[https://www.cs.swarthmore.edu/~newhall/unixhelp/howto\\_makefiles.html](https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html)

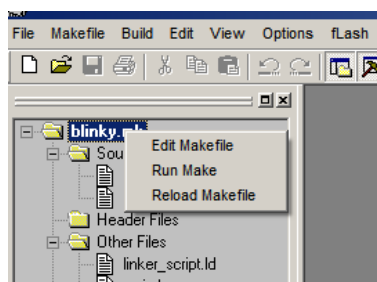
<https://en.wikipedia.org/wiki/Makefile>

## 2. Using Makefiles with CrossIDE: Compiling, Linking, and Loading.

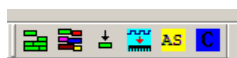
To open a Makefile in CrossIDE, click “Makefile”→”Open” and select the Makefile to open. For example “blinky.mk”. The project panel is displayed showing all the targets and source files:



Double clicking a source file will open it in the source code editor of CrossIDE. Double clicking a target ‘makes’ that target. Right clicking the Makefile name shows a pop-up menu that allows for editing, running, or reloading of the Makefile:



Additionally, the Makefile can be run by means of the Build menu or by using the Build Bar:



Clicking the ‘wall’ with green ‘bricks’ makes only the files that changed since the last build. Clicking the ‘wall’ with colored ‘bricks’ makes all the files. Clicking the ‘brick’ with an arrow, makes only the selected target. You can also use F7 to make only the files that changed since the last build and Ctrl+F7 to make only the selected target.

### Compiling & Linking

After clicking the build button this output is displayed in the report panel of CrossIDE:

```
----- CrossIde - Running Make -----
del init.o          main.o
del main.elf
arm-none-eabi-gcc -c -mcpu=cortex-m0 -mthumb -g init.c -o init.o
arm-none-eabi-gcc -c -mcpu=cortex-m0 -mthumb -g main.c -o main.o
arm-none-eabi-ld init.o main.o -T lpc824_linker_script.ld --cref -nostartfiles -Map main.map
-o main.elf
objcopy -O ihex main.elf main.hex
```

### Loading the Hex File into the Microcontroller’s Flash Memory

In order to load the hex file into the LPC824’s flash memory, the serial boot loader<sup>3</sup> needs to be activated first. To activate the boot loader, press and hold the Boot push button, then press and release the Reset push button, and finally release the Boot push button. Once the boot loader is activated double click the ‘FlashLoad’ target. This output is then displayed in the report panel of CrossIDE:

```
----- CrossIde - Running Make -----
lpcprog.exe main.hex -FindPort 115200 12000000
lpc2lisp version 1.97
File main.hex:
    loaded...
    converted to binary format...
    image size : 548
Image size : 548
Searching ports.
LPC microcontroller found at COM128
Synchronizing (ESC to abort)..... OK
Read bootcode version: 1.13.0
Read part ID: LPC824M201JDH20, 32 kiB FLASH / 8 kiB SRAM (0x00008242)
Will start programming at Sector 1 if possible, and conclude with Sector 0 to ensure that
checksum is written last.
Erasing sector 0 first, to invalidate checksum. OK
*
Download Finished... taking 0 seconds
Now launching the brand new code
```

A file named “COMPORT.inc” is created after running the flash loader program. The file contains the name of the port used to load the program, for example, in the example above COM128 is stored in the file. “COMPORT.inc” can be used in the Makefile to create a target that starts a PuTTY serial terminal session using the correct serial port:

```
PORTN=$(shell type COMPORT.inc)
.
.
putty:
    @Taskkill /IM putty.exe /F 2>NUL | wait 500
    c:\putty\putty.exe -serial $(PORTN) -sercfg 115200,8,n,1,N -v
```

For more details about using “COMPORT.inc” check the project examples below.

---

<sup>3</sup> The boot loader comes preloaded into the LPC824 microcontroller from factory.

## Setting up the Development Environment on macOS

Download and install Visual Studio Code (VS Code) for macOS from:

<https://code.visualstudio.com/Download>

In VS Code install the extension “Make support and task provider” by carlos-algms.

Install xcode and homebrew for macOS. I used the instructions from this link:

<https://phoenixnap.com/kb/install-homebrew-on-mac>

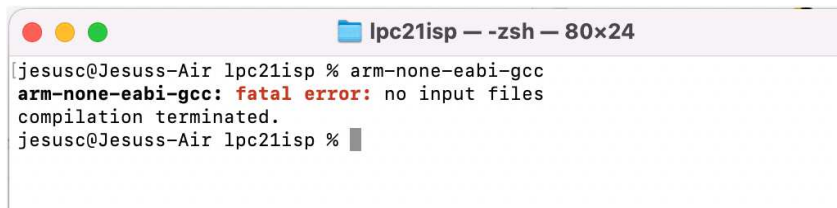
Install arm-none-eabi-gcc<sup>4</sup> using homebrew. In a terminal type:

```
$ brew tap ArmMbed/homebrew-formulae
$ brew install arm-none-eabi-gcc
```

To test that the ARM compiler is installed correctly, type in a terminal the following:

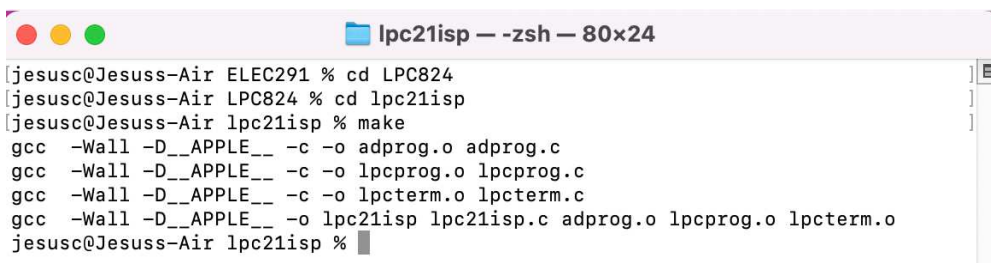
```
$ arm-none-eabi-gcc
```

The program runs and complains that there are no source files to compile (this is a good thing because it means that the compiler is now installed!):



```
lpc21isp — -zsh — 80x24
[jesusc@Jesuss-Air lpc21isp % arm-none-eabi-gcc
arm-none-eabi-gcc: fatal error: no input files
compilation terminated.
jesusc@Jesuss-Air lpc21isp %
```

Download the project examples for the LPC824 from the course web page (Canvas) and unzip it somewhere in the hard drive of your Mac computer. The flash loader included with the zip file for the LP824 has to be re-compiled for macOS. In a shell terminal navigate to the directory lpc21isp and type ‘make’. It looks like this:



```
lpc21isp — -zsh — 80x24
[jesusc@Jesuss-Air ELEC291 % cd LPC824
jesusc@Jesuss-Air LPC824 % cd lpc21isp
jesusc@Jesuss-Air lpc21isp % make
gcc -Wall -D__APPLE__ -c -o adprog.o adprog.c
gcc -Wall -D__APPLE__ -c -o lpcprog.o lpcprog.c
gcc -Wall -D__APPLE__ -c -o lpcterm.o lpcterm.c
gcc -Wall -D__APPLE__ -o lpc21isp lpc21isp.c adprog.o lpcprog.o lpcterm.o
jesusc@Jesuss-Air lpc21isp %
```

Each LPC824 project example should include a ‘makefile’ for macOS (if not you can make one based on the ‘makefile’ for Windows). The file is named ‘makefile.mac’. For example, for project ‘Blinky’, the file ‘makefile.mac’ looks like this:

<sup>4</sup> I followed the instructions here

<https://dev.to/lewuathe/how-to-achieve-arm-cross-compilation-on-macos-3b08>



```

CCFLAGS=-mcpu=cortex-m0 -mthumb -g -lgcc
PORTN=/dev/$(shell ls /dev | grep "cu.usbserial")
OBS=init.o main.o

# Search for the path of libraries.
LIBPATH1=$(shell find /opt -name libgcc.a | grep "v6-m" | sed -e "s/libgcc.a/g")
LIBPATH2=$(shell find /opt -name libc_nano.a | grep "v6-m" | sed -e "s/libc_nano.a/g")
LIBSPEC=-L"${LIBPATH1}" -L"${LIBPATH2}"

main.elf : $(OBS)
    $(LD) $(OBS) $(LIBSPEC) -lgcc -T lpc824_linker_script.ld --cref -Map main.map -o main.elf
    arm-none-eabi-objcopy -O ihex main.elf main.hex
    @echo Success!

main.o: main.c lpc824.h
    $(CC) -c $(CCFLAGS) main.c -o main.o

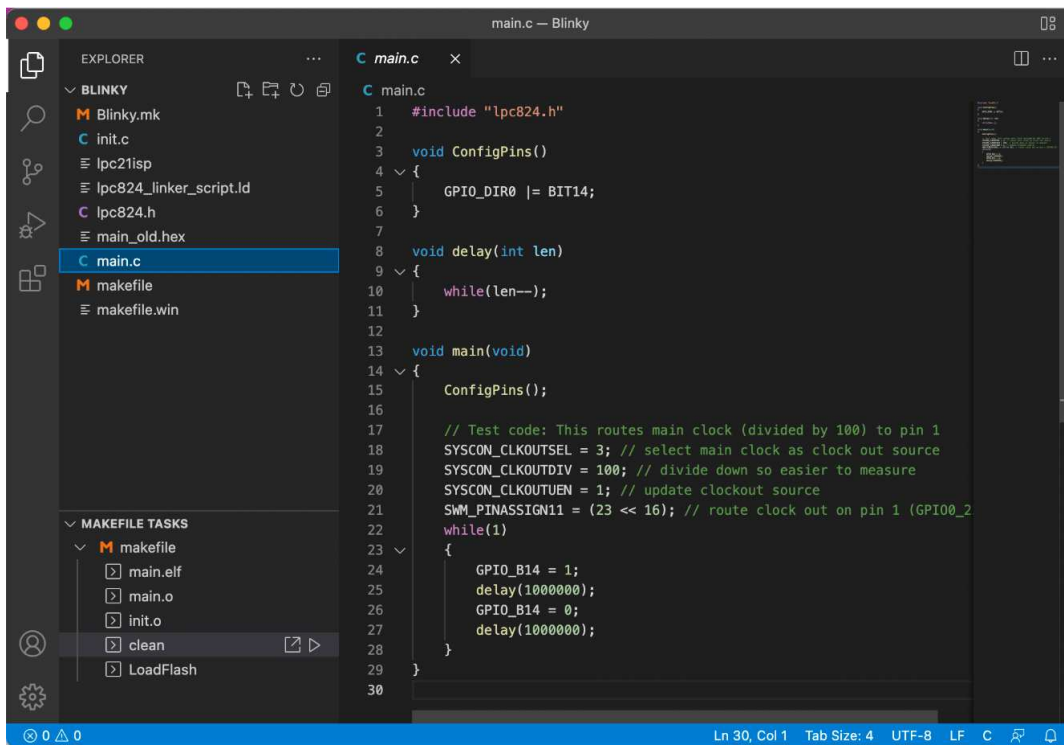
init.o: init.c lpc824.h
    $(CC) -c $(CCFLAGS) init.c -o init.o

clean:
    rm -f $(OBS) main.elf main.hex main.map

LoadFlash:
    ./lpc21isp/lpc21isp main.hex $(PORTN) 115200 12000000

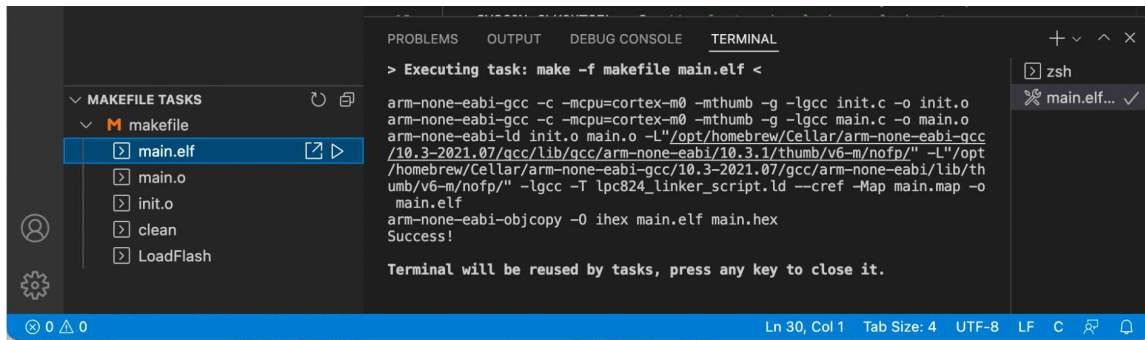
```

Rename the file ‘makefile.mac’ to ‘makefile’. Open the ‘Blinky’ directory in VS Code. It should look something like this:



To compile and link the program, under ‘MAKEFILE TASKS’ click ‘main.elf’:

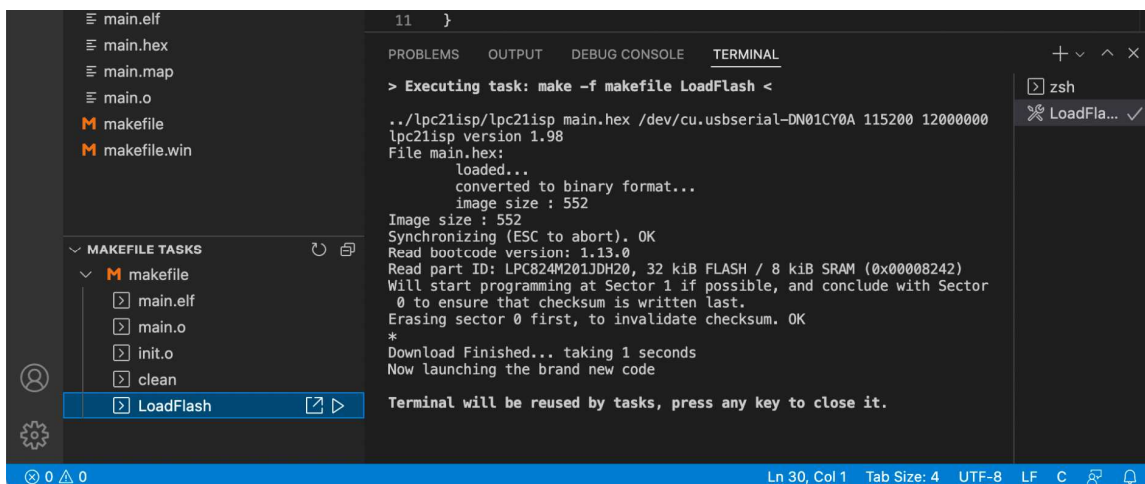




To load the new program into the LPC824, first you'll need to activate the serial boot loader:

1. Press and hold the BOOT push button.
2. Press and release the RESET push button.
3. Release the BOOT push button.

Now, under 'MAKEFILE TASKS' click 'LoadFlash':



That is all. The program should be running in the LPC824.

## Project Examples

The following Project examples are available in web page of the course. Some of these projects and files are based on examples posted by Frank Duignan in:

<http://eleceng.dit.ie/frank/arm/BareMetalLPC824/index.html>

**Blinky:** ‘blinks’ an LED connected to pin GPIO\_B14. This is the same project used in the examples above.

**HelloWorld:** Uses printf() to display “Hello, World!” using the serial port. You need to modify the Makefile for this project to reflect the location of the required libraries as per the “LIBSPEC” assignment.

**MRTimer:** Similar to “Blinky” but instead of using a delay loop, it uses a Multi-rate timer interrupt. Two LEDs are made to blink at different rates using two different timers. The LED connected to GPIO\_B14 blinks at a rate of 2 Hz. The LED connected to GPIO\_B15 blinks at a rate of 1 Hz.

**SCTimer:** Similar to “Blinky” but instead of using a delay loop, it uses the State Configurable Timer (SCT) interrupt. Two LEDs are made to blink at different rates similarly to the previous project.

**PrintFloat:** Shows how to print numbers using printf().

**PrintADC:** Reads a channel of the built in ADC (ADC[3], pin 1) and displays the result using printf() and PuTTY. Since this project uses printf() with floating point support, a significant amount of flash memory is used: 23917 bytes<sup>5</sup>.

**PrintADCEff1:** Reads a channel of the built in ADC (ADC[3], pin 15) and displays the result using printf() and PuTTY. This project does not use floating point with printf(); therefore the amount of flash memory used is about 1/3 the size of the previous project: 8987 bytes.

**PrintADCEff2:** Reads a channel of the built in ADC (ADC[3], pin 15) and displays the result using the serial port and PuTTY. This project does not use printf(); therefore the amount of flash memory used is about 1/4 the size of the previous project: 2373 bytes.

**Freq\_Gen:** Shows how to generate an arbitrary frequency square wave at one of the pins of the microcontroller.

**Servo:** Shows how to generate the standard servo motor controller signal.

---

<sup>5</sup> The LPC824 has 32768 bytes of Flash memory.