Experiment 1: Development Platform

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
141	Sep. 7, 2014		A

Contents

1 Objectives **1**

2 Parts List **1**

3 Background **1**

3.1 Microcontroller . . . 1

3.2 Development Board . . . 2

3.3 LPCXpresso IDE . . . 2

3.3.1 Installation . . . 2

3.3.2 Activation . . . 3

3.4 Input/Output Ports . . . 3

3.4.1 Memory-Mapped I/O . . . 3

3.4.2 General-Purpose Input/Output (GPIO) . . . 4

3.4.3 Naming Registers . . . 4

3.5 Bit Manipulation in C . . . 4

3.5.1 Bitwise Operators . . . 4

3.5.2 Masking . . . 5

3.5.3 Creating Masks by Shifting . . . 5

4 Tasks **6**

4.1 Create a Project . . . 6

4.2 Blink an LED . . . 6

4.3 Debug Your Project . . . 6

5 Resources **7**

1 Objectives

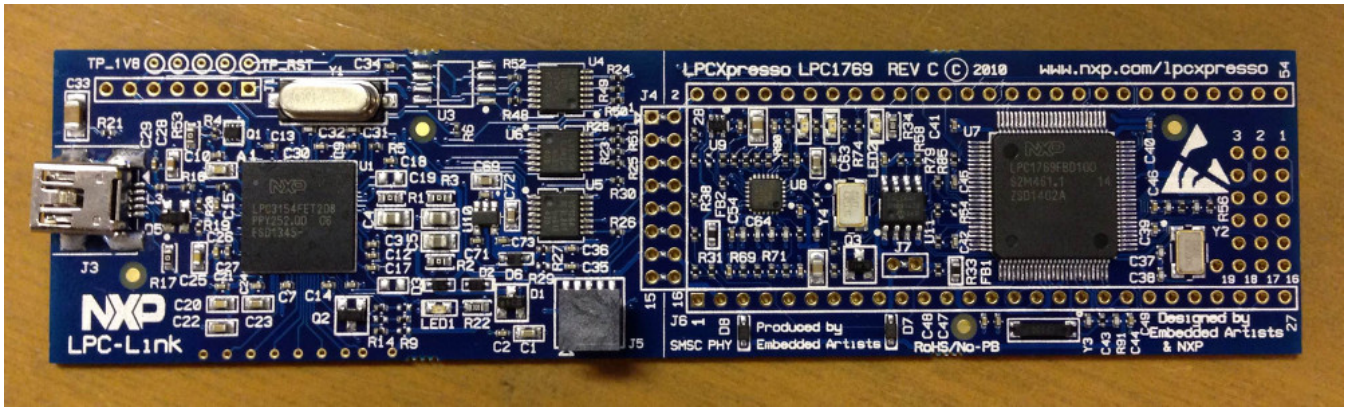
- Get familiar with the development platform:

Hardware	microcontroller, development board, peripherals
Software	IDE, compiler, debugging, programming

- General-purpose input/output (GPIO): digital output

2 Parts List

- LPC1769 LPCXpresso board



- USB A-Type to Mini-B cable

3 Background

3.1 Microcontroller

LPC1769 is a microcontroller manufactured by NXP. NXP was founded by Philips as Philips Semiconductors, and renamed NXP in 2006. A major difference between a microcontroller and a microprocessor is that the former has some additional built-in

devices, such as memory, I/O peripherals, and timers.

The LPC1769 microcontroller is an ARM 32-bit Cortex-M3 microcontroller. Some of its features include: CPU clock up to 120MHz, 512kB on-chip Flash ROM, 64kB RAM, Ethernet 10/100 MAC, USB 2.0 full-speed Device controller and Host controller, four UARTs, general purpose I/O pins, 12-bit ADC, 10-bit DAC, four 32-bit timers, Real Time Clock, System Tick Timer.

The product data sheet for the LPC1769 microcontroller [lpc1769] is an essential resource for any developer.

3.2 Development Board

The LPC1769 microcontroller chip is used to build the *LPC1769 LPCXpresso board*, which we will be using in this LAB.

The LPCXpresso board consists of two parts:

1. LPCXpresso target board, which hosts the LPC1769 microcontroller
2. LPC-Link: a debug probe for debugging and the target microcontroller.

For an overview of the LPCXpresso platform (hardware and software), see [lpcx-platform].

3.3 LPCXpresso IDE

The LPCXpresso IDE is an Eclipse-based software development environment for NXP's LPC microcontrollers.

The LPCXpresso IDE uses the GNU toolchain (compiler and linker), and offers the choice of two C libraries:

- Redlib (default): a proprietary ISO C90 standard C library, with some C99 extensions. Often results in smaller binary size.
- Newlib: an open source complete C99 and C++ library.

The LPCXpresso IDE is available in two editions:

- Free edition: requires free activation (details below). Supports code sizes up to 256 kB after activation.
- Pro edition: per-seat licensing fees. Supports unlimited code size.

3.3.1 Installation

To install your copy of the LPCXpresso IDE:

1. Using a web browser, navigate to the download page:
   http://www.lpcware.com/lpcxpresso/download
2. Under *Product downloads*, follow the link that matches your operating system. LPCXpresso supports Windows, Linux, and Mac OS X.

   Note
   You may also want to get a copy of the *LPCXpresso 7 User Guide* listed at the top of the page.

3. Click on the first download link in the download page for your operating system. The link text looks like:

   ```
   Download the current release (7.x.y) of LPCXpresso 7 for ...
   ```

4. Run the downloaded installer.

3.3.2 Activation

To activate your copy of the LPCXpresso IDE:

1. Using a web browser, log into the activation web page:
   http://www.lpcware.com/lpcxpresso/activate
   Register a new account if you do not have one.
2. In the LPCXpresso IDE, select Help > Product Activation > Create Serial number and Activate.
3. Copy the shown serial number, and paste it into the activation web page from step 1.
4. You will receive an *LPCXpresso Key Activation* email, which includes the *Product Activation Key*.
5. In the LPCXpresso IDE, select *Help* > *Enter Activation Code*, and copy the *Product Activation Key* received in the email into this form.

For more information on installing and using the LPCXpresso IDE, see [lpcx-ide].

3.4 Input/Output Ports

The LPC1769 microcontroller has five input/output (I/O) ports. Each port is 32 bits. However, not all of them are available for the developer. For example, pins 12, 13, 14, and 31 of port 0 are not available, leaving only 28 pins available for the user.

Each of the I/O pins is referred to using the port number and the pin number. For example, P0.17 or P0[17] is pin 17 in port 0, and P1.22 or P1[21] is pin 22 in port1. In some cases, the same pin is called by its actual pin number on the chip; For example, P0.17 is called pin-61 and P1.22 is called pin-35.

Most of the I/O pins have multiple functions. For example: P0.10 can perform one of these jobs:

P0[10]	General purpose digital input/output pin.
TXD2	Transmitter output for UART2.
SDA2	I2C2 data input/output.
MAT3[0]	Match output for Timer 3, channel 0.

Note
Don't worry if you don't understand these functions, you will learn about them throughout the course.

A command is needed to choose which function is to be used in a specific pin. The only exception is the first function (GPIO) because it is the default function.

Note
Relying on a default value may be acceptable in simple programs. However, a good programming style when you have many functions and interrupts is not to assume any default value as they may have been changed somewhere in your program. Instead, you should explicitly specify any desired values.

3.4.1 Memory-Mapped I/O

ARM uses memory-mapped I/O. When using memory-mapped I/O, the same address space is shared by memory and I/O devices. Some addresses represent memory locations, while others represent registers in I/O devices. No separate I/O instructions are

needed in a CPU that uses memory-mapped I/O. Instead, we can use any instruction that can reference memory to move values to or from memory-mapped device registers.

3.4.2 General-Purpose Input/Output (GPIO)

GPIO is available in most I/O pins. A GPIO pin is a pin that can be used for digital input or digital output. Clearly you need a command (instruction) to choose the direction of the pin (whether it is used for input or output). In the first example of this experiment we will set the direction to GPO (General Purpose Output). To use a digital output pin, you need a command (instruction) to *set* the output to HIGH (1), and a command to *clear* it to LOW (0).

In summery, we need to learn about 3 registers for our first experiment:

1. The register that controls the direction of GPIO pins
2. How to *set* that pin to HIGH.
3. How to *clear* that pin to LOW.

3.4.3 Naming Registers

Each I/O register has an address. For example:

1. The address of the register that controls the direction of port 0 pins is: `0x2009c000`.
2. The address of the register that *sets* port 0 pins to HIGH is: `0x2009c018`.
3. The address of the register that *clears* port 0 pins to LOW is: `0x2009c01c`.

One way to give these registers some names is as follows:

```
#define DIR_P0  (*((volatile unsigned long *) 0x2009c000))
#define SET_P0  (*((volatile unsigned long *) 0x2009c018))
#define CLR_P0  (*((volatile unsigned long *) 0x2009c01c))
```

3.5 Bit Manipulation in C

The core of embedded system programming is setting (or clearing) specific bits in different registers inside the microcontroller. This highlights the importance of *bit manipulation* as a programming skill.

Most modern architectures are byte-addressable: the smallest unit of data is the byte. Nonetheless, it is possible to operate on individual bits by clever use of bitwise operators.

3.5.1 Bitwise Operators

Bitwise operators apply to each bit of their operands.

Operator	Function	Examples
`&`	Bitwise AND	`0011 & 0101 = 0001` `3 & 5 = 1`
`|`	Bitwise OR	`0011 | 0101 = 0111` `3 | 5 = 7`

Operator	Function	Examples
^	Bitwise XOR	`0011 ^ 0101 = 0110` `3 ^ 5 = 6`
~	Bitwise NOT	`~00110101 = 11001010`
<<	Shift left	`3 << 2 = 12`
>>	Shift right	`8 >> 2 = 2`

Note
In C, numbers can be written in decimal, octal, or hexadecimal, but not in binary, e.g. `16 =020 =0x10`.

Warning
Right-shifting in C is implementation-specific. Often, *logical shifting* is applied to unsigned values, whereas *arithmetic shifting* is applied to signed values.

3.5.2 Masking

A simple assignment to a 32-bit register or memory location will overwrite all 32 bits. However, manipulating specific bits implies that the remaining bits in the register remain intact. An essential technique to achieve that is bit masking.

A mask is a value that can be used in a binary, i.e. two-operand, bitwise operation to change specific bits of some other value. Masking relies on the following rules of Boolean Algebra:

- ANDing a bit with a `0` results in a `0`. ANDing a bit with a `1` results in the same bit.
- ORing a bit with a `0` results in the same bit. ORing a bit with a `1` results in a `1`.
- XORing a bit with a `0` results in the same value. XORing a bit with a `1` inverts the bit.

EXERCISES

1. What mask and bitwise operation are required to set the third least significant bit (bit 2) to `1` without affecting the other bits in a 32-bit variable `x`?
2. What mask and bitwise operation are required to reset bit 10 of a 16-bit variable `y` to `0`?
3. What mask and bitwise operation are required to toggle bit 20 of a 32-bit variable `z`?

3.5.3 Creating Masks by Shifting

If you have worked out the exercises above, you would have noticed that spelling out masks can be tedious, verbose, and error-prone. One trick that makes it easier to create masks is to use the shift operations. For example, to create a mask whose bit 10 is `1` and whose other bits are `0`, you can use the following C statement:

```
mask = 1 << 10;
```

Exercises Repeat the three exercises above by using shift operations to create the masks.

4 Tasks

4.1 Create a Project

1. Click *Quickstart Panel* > *New project...*.
2. Choose *LPC13 / LPC15 / LPC17 / LPC18* > *LPC175x_6x* > *C Project*. Click *Next*.
3. Choose a project name, e.g. `cmsis_blinky`, and click *Next*.
4. In the *Target selection* dialog, choose *LPC1700* > *LPC1769*.
5. In the *CMSIS Library Project Selection* dialog, set *CMSIS Core library to link project to* to `None`, then click *Next*.
6. In the *CMSIS DSP Library Project Selection* dialog, set *CMSIS DSP Library to link project to* to `None`, then click *Next*.
7. Go to `main` function and start writing your code.
8. Uncheck *Enable linker support for CRP*, then click *Finish*.

4.2 Blink an LED

To blink an LED, you need to do the following:

1. Figure out which pin is connected to the LED.

   Tip
   Refer to the LPC1769 board documentation.

2. Give the required registers some friendly names using the above `#define` directives.
3. In an infinite loop inside the `main` function:
   a. Set the pin to act as output by setting the correct bit in the direction register to 1.
   b. Set the output pin to 1.
   c. Clear the output pin (set to 0).
   d. Insert a delay loop after both set and clear, to be able to see the LED blink.
4. Which value of the pin turns the LED on, and which value turns it off? and why?

4.3 Debug Your Project

1. Click *Quickstart Panel* > *Build* cmsis_blinky *[Debug]* to build the project.
2. Connect the LPC1769 board to the PC using the USB cable.
3. Click *Quickstart Panel* > *Debug* cmsis_blinky *[Debug]* to debug the project interactively on the target board.

   Running the Debugger
   You can run the debugger using any of the following three ways:

   1. In the *Quickstart Panel* at the lower left corner, click *Debug* <project-name> *[Debug]*.
   2. In the main menu, choose *Run* > *Debug As* > *C/C++ (NXP Semiconductors) MCU Application*.
   3. In the toolbar, click on the debug button  .

4. Once the debugger starts, it will pause execution at the first statement in the progrm. Resume execution by hitting the `F8` key, or using the resume button in the toolbar  .

5 Resources

[lpc1769]
NXP Semiconductors. *LPC1769/68/67/66/65/64/63 — Product data sheet*. Rev. 9.5. 24 June 2014.
http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf

[lpcx-platform]
NXP Semiconductors. *Getting started with NXP LPCXpresso (User guide)*. Rev. 12. 17 April 2013.
http://www.nxp.com/documents/other/LPCXpresso_Getting_Started_User_Guide.pdf

[lpcx-ide]
NXP Semiconductors. *LPCXpresso v7 User Guide*. Rev. 7.3. 30 June 2014.
http://www.lpcware.com/system/files/LPCXpresso_User_Guide_0.pdf