Lab D: Getting Started with your ARM µC's GPIO

Introduction

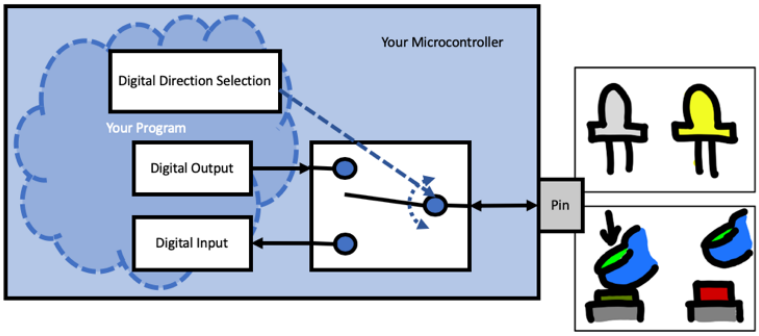
Single Sentence Overview

Learn how to use your microcontroller ("µC"), as well as LED lights and a switch attached to it.

Overview of Lab 1

This three-part lab is designed to introduce the student to a debugging-centric method for developing a program on a microcontroller.

- **First**, the student will write a simple program, download it onto the board and verify that it is working.
- **Second**, the student will use the debugger memory access functions to examine whether a button is being pressed and whether the LEDs can be made to turn on.
- **Third**, the student will write a program in C that turns an LED on and off.



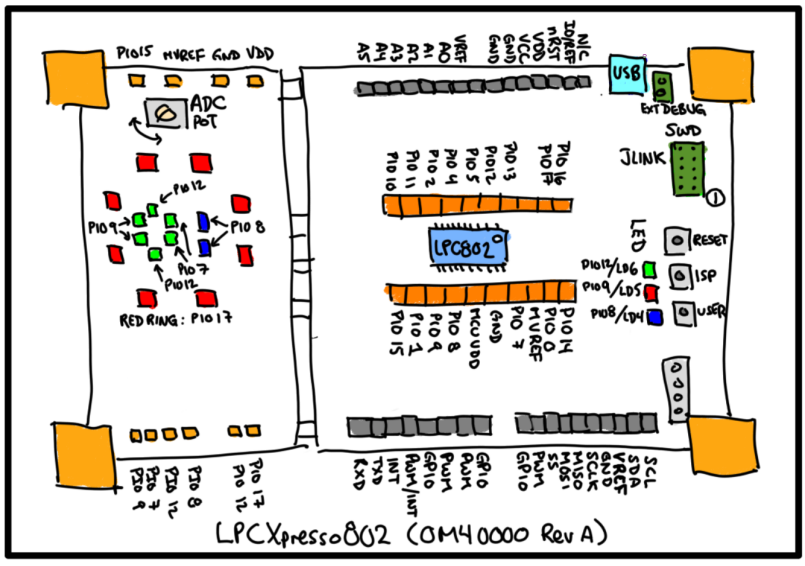


Figure 1 You will use a USB connection to your computer and the internal debugger circuitry to access the internal memory of the NXP LPC802 microcontroller, as well as a switch and LED lights attached to it.

LPC 804	Note: if you are using the LPC804 (OM40001 board), text boxes will be placed in appropriate parts of this document to signal important differences.

Learning Outcomes of Lab D

The student will know how to

1. Write a generic program in C and make sure it works without accessing (much) special chip-specific features.
2. Use the debugger to manually access the inner features (registers) of the microcontroller, as well as the LEDs and Switch on your board.
3. Write a chip-specific program in C that can turn on LEDs on the board using the debug "step" feature.

Success Criteria

Review the grading rubric and evaluate how it relates to achieving these criteria:

1. Write a simple program in C and demonstrate that it works on the board.
2. Show that you can use the debugger to manually read a digital input (switch) and set a digital output (LED).
3. Demonstrate a C program that can flash an LED repeatedly

Grading Rubric

During the lab session make sure to conduct all the required demonstrations to the lab instructor.

- *Part 1 demo: no grade*
- *Part 2 demo:*
  - *0 pts: no demonstration attempted or neither worked at all.*
  - *5 pts: One of the two demonstrations worked, or both worked only partly.*
  - *10 pts: Both demonstrations worked (LED and pushbutton switch)*
- *Part 3 demo:*
  - *0 pts: no demonstration attempted.*
  - *5 pts: LED only partially worked or was not fully implemented in C and / or in debug mode.*
  - *10 pts: LED turned on / off using C in debug mode.*

There is **no requirement to submit a document** with the answers to the "Stimulate" and "Explore" questions. Those questions are there for you to reflect upon.

Prerequisites

You are expected to have done the following before attending the lab session:

- Have attended class
- Consider watching this video: https://youtu.be/Fgg5ONNHZrk

More Resources and Information

- The LPC802 board (OM40000) User Manual (available on Moodle)
- The LPC802 (OM40000) board schematic (available on Moodle)

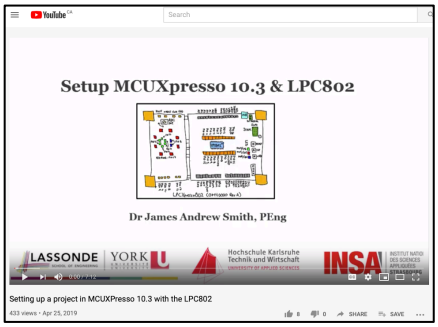
LPC 804	The LPC804 board (OM40001) and schematics are available on the Moodle site, as well.

Part 0: Getting started.

There are some details that are worth noting when using MCUXPresso for the first time.

A note about running MCUXpresso in the Lassonde computer labs (e.g. GSP lab).

1. From the Linux screen, hit the Windows key on the keyboard and type in Windows. Reboot into Windows
2. In Windows, open up MCUXpresso **11** (not 10).
3. When prompted, set your MCUXpresso workspace to be in your user directory on Z: drive (it's the default)
    a. If you put it on C: it will get erased when you reboot.
4. Drag the SDK file for the LPC802 or 804 from C:\nxp\mcuxpresso\board-support-packages to the bottom of the MCUXpresso window (SDK area)
5. When making a new project, choose File->New->Project->C/C++->New C/C++ Project
    a. Alternatively, use "New Project" in the Quickstart Panel (bottom left of IDE)
6. Choose the LPC802 or LPC804 with the picture that says "SDK" on it.
7. When the project initializes, go into the main source code file, erase all the contents in that file and write everything from scratch (as per the lab instructions)
    a. The source file is found in the Project Explorer panel, on the left.
    b. Look for your project name & then the "source" folder. Open it.



A setup video with MCUXpresso & the LPC802: https://youtu.be/Fgg5ONNHZrk

Part 1: Introduction to Debugging

Figure 2 Debuggers are like Star Trek's Vulcans. They can "mind-meld" with your chip.

One Sentence Summary

Learn to use the board's built-in debugger to turn on an LED.

Overview

In the first part of your lab, you will get familiar with the hardware for your LPC802 microcontroller and the software we use to write software for the LPC802.

Learning Objectives

The general learning objective is to show you that programming for embedded devices*)* like the LPC802 *can* and *should* be done from a debugger-centric perspective. This method is compatible with the goal of self-directed learning that is typical of many engineering programs.[1]

In the process of learning to develop from a "debugging-centric" perspective, you will explore the tools for developing programs on a microcontroller, including:

- The LPC802 chip
- The board the chip sits on
- The built-in debugger that connects to the board to your computer and, finally,
- the MCUXpresso software for writing programs for the LPC802.

Success Criteria

When you have completed this learning module you will be able to demonstrate that you can write a simple program in C and run a debugging session that connects to the LPC802 on your board.

The Main Topic: The Debugger as your access into the Microcontroller.

Debuggers[2] like the one built-in to your development board (see **Error! Reference source not found.**) allow you to examine the inner workings of your microcontroller. They are professional-grade tools and allow you to

- Program your chip
- Read existing programs or data on the chip
- Change settings manually on the chip
- Analyze behaviour of your program.

Debuggers are widely used by embedded systems programmers (and students!) to ensure that the programs they write and the hardware they design work as intended. They are also used by security professionals to find weaknesses in electronic systems.

[1] Labs should encourage structured and facilitated, but self-directed learning. To this end, this document is designed around a modified version of the Process-Oriented Guided Inquiry Learning (POGIL) model, and is compatible with the North American ABET and CEAB models for engineering education. A good applied example of this process is in the "Laboratory Short Course" series produced for Freescale's 9s12 processors by Fred Cady, Natasha Kholgade and Ken Hsu. Dr. Cady has given permission to modify his original material.

[2] External debuggers like the Segger JLink (for ARM and FPGA) or PICKit 4 (for Microchip products) are inexpensive and can be used when your board doesn't have one built in. (https://binged.it/2v9iq9F and https://youtu.be/Bhd8644wvf8?t=386)

Writing your first program

Your first program won't do much, but it will allow you to connect your LPC802 to your computer. Here are the steps to follow to get **started**

1. Connect your development board to your PC.
2. Open MCUXpresso & start a New Project (then New -> Project)
3. Select the SDK for the LPC802 board
4. Use the defaults and call your project LabD_version1

Now that you've got a project started, you need to start **writing your C program**:

1. After the main project window appears, double-click on the "source" folder
2. Double-click on the "LabD_version1.c" file
3. Open up the resulting C file
4. Modify the C file
    a. See the image to the right
5. Make sure that your project is selected as the "main" one. That is, make sure to "close unrelated projects"
6. **Compile** your project with the Hammer icon.

```
// EECS 3215 Lab D, part 1
#include "LPC802.h"
int main(void) {
  // Turn on the GPIO system.
  SYSCON->SYSAHBCLKCTRL0 |= SYSCON_SYSAHBCLKCTRL0_GPIO0_MASK;
  // loop infinitely
  while(1)
  {
      asm("NOP");
  }
} // end of main
```

If your program compiled, then move on to the **debugging** stage.

1. Make sure that you have connected the development board to your PC.
2. Right-click on the project name in the Project Explorer pane
3. Click on Debug As -> MCUXpresso IDE Linkserver
    a. (later you can use the blue or green **bug** icon shortcut)
4. Confirm that you are using the Linkserver when this window appears
5. The bottom of your window should be very active with text now. The program is recompiling and being sent to the board
6. You will be told that your program has halted. It is now paused at the first line of your main function. Good.

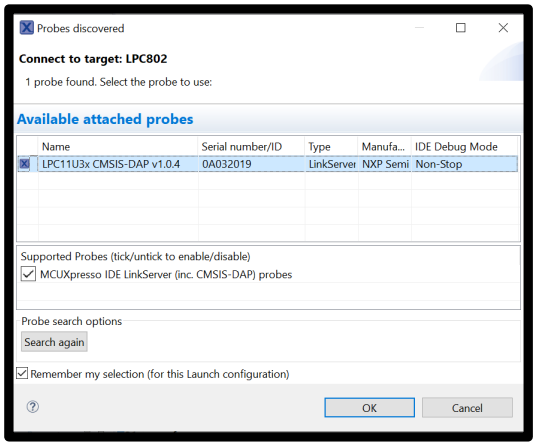


Figure 3 The Windows "defender" may object to this. You either need to wait for Defender to go away or to give the debugger permission to flow through the firewall.

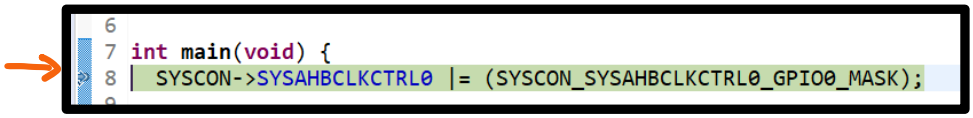


Figure 4 By default, when the program starts in debug mode it halts at the first line inside the main function. Here, that's line 8. The halting point is indicated by the green arrow beside the 8.

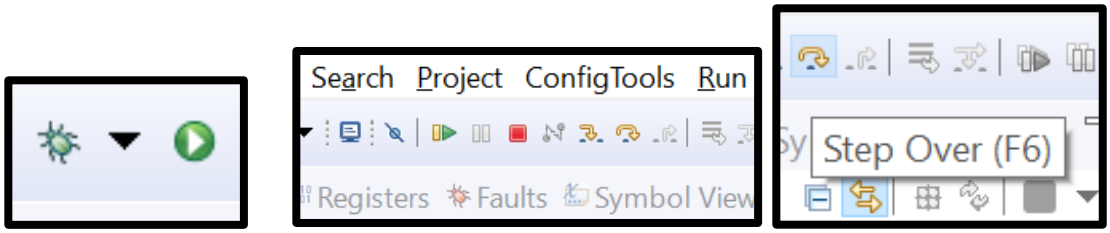


Figure 5 The Debug Project engages the debugger (left icon). When you add a breakpoint (red box, middle image), the debugger will halt on it (green arrow, top image), allowing you to use the yellow arrows to "Step over" (right icon) or "Step into" through your code.

That is all you need to do at this point. You're simply confirming that you can access the board using the debugger.

There is nothing to submit for this portion of the lab. Just make sure that it works and ask for clarification if it does not or does not work as you thought it might.

Part 2: Directly Accessing the Microcontroller's Resources

Single Sentence Overview

You will access the internal resources directly on the microcontroller with almost no programming required.

Overview

The debugger is a powerful tool that allows you to not only program the chip but also to monitor and change resources inside the chip. Here we'll see how the Special Function Register View allows us to modify the values and functions of the chip's pins without writing commands in C or Assembler. This approach can be used with more complex programming problems to verify that your program is executing as intended and, if it is not, why.

Background material

The debugger is capable of directly examining and setting up the hardware on your LPC802 chip. This is useful when either exploring your hardware for the first time or for understanding the chip's behaviour when something doesn't go right at a later stage. To get started, you should generally look at two documents:

1. The board's schematic
2. The chip's datasheet (or user manual)
    a. The LPC802's "user manual" is more comprehensive than its datasheet. Use the user manual.

The board schematic helps narrow down the chip's pins that are of direct importance to you. You then use the labels to search through the chip datasheet for hints as to which registers need direct manipulation by the debugger. Usually, these are:

1. Port input value register (e.g. GPIO->B[0][8])
2. Port output set or clear value registers (e.g. GPIO->SET[0] = (1UL<<LED_USER2); or GPIO->CLR[0] = (1UL<<LED_USER2);
3. Data Direction register (e.g. GPIO->DIRSET[0] = (1UL<<LED_USER1); or GPIO->DIRCLR[0] = (1UL<<BUTTON_USER1);)

You can do all of these steps in Assembler programming, C programming, direct debugger register manipulation, or a combination of any of these three. Here, we will do this directly via the debugger and a simple C program.

Explore

1. When the green debug arrow arrives at a line in your C code, does it mean that
    a. The code ran in the past?
    b. The code is running in the present?
    c. The code will run in the future?
2. Can you get this code to compile and execute using the C++14 (ISO) compiler in MCUXpresso?
    a. Were any changes required to the code?

The Problem

Write a simple program as you did in Part 1, with a main function that contains only these lines:

```
// EECS 3215 Lab D, part 2
#include "LPC802.h"
int main(void) {
  // Turn on the GPIO system.
  SYSCON->SYSAHBCLKCTRL0 |= SYSCON_SYSAHBCLKCTRL0_GPIO0_MASK;
  // loop infinitely
  while(1)
  {
      asm("NOP");
  }
} // end of main
```

LPC 804	Use the “LPC804.h” header file, instead.

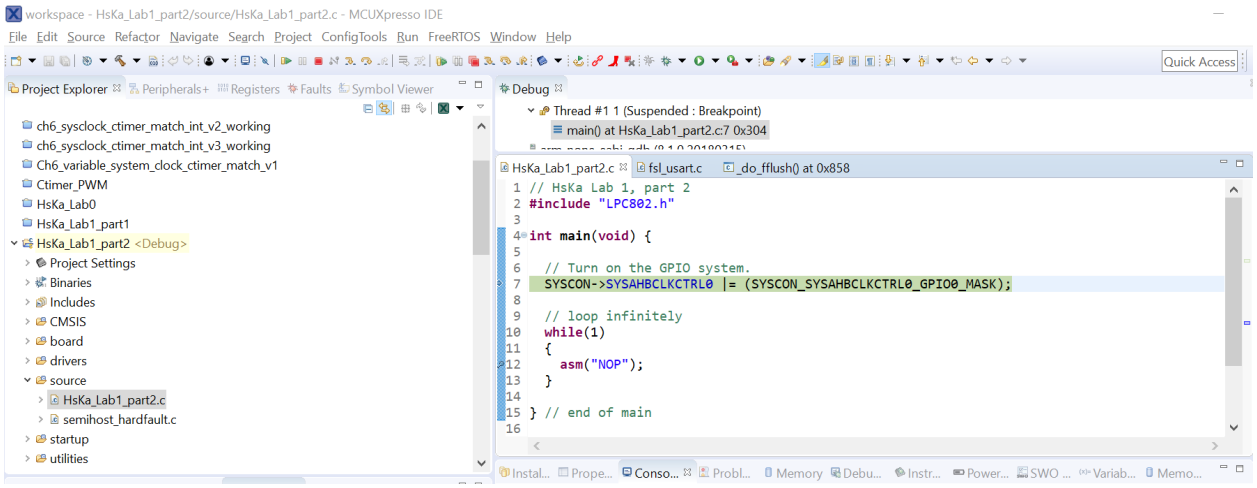


Figure 6 With the program halted at the breakpoint we can modify the peripheral registers, as well as monitor incoming signals using the Peripherals view. Note that MCUXpresso defaults to halting on the first line in main. Add a breakpoint to the asm(“NOP”) line, too, by double-clicking on the line number (12 in this case).

Compile the program to ensure that it is valid C code.[3] Then add a breakpoint to the `NOP` line and run the code on your board using the debug mode in MPLAB X. The code will execute on your board and halt at the `NOP` line. You can now *examine or modify* the internal settings of the microcontroller without running

[3] Valid C code… but it’s also valid C++ code (I’ve tested it against the ISO C++14 compiler in MCUXpresso 11)

any more C or Assembler program code. You can access each register inside the chip and modify the behaviour of the chip directly.

Here you will use MCUXpresso's **Peripherals panel** and its ability to modify the configuration of the pins and then modify the voltage output of the pins so that the LEDs turn on and off.

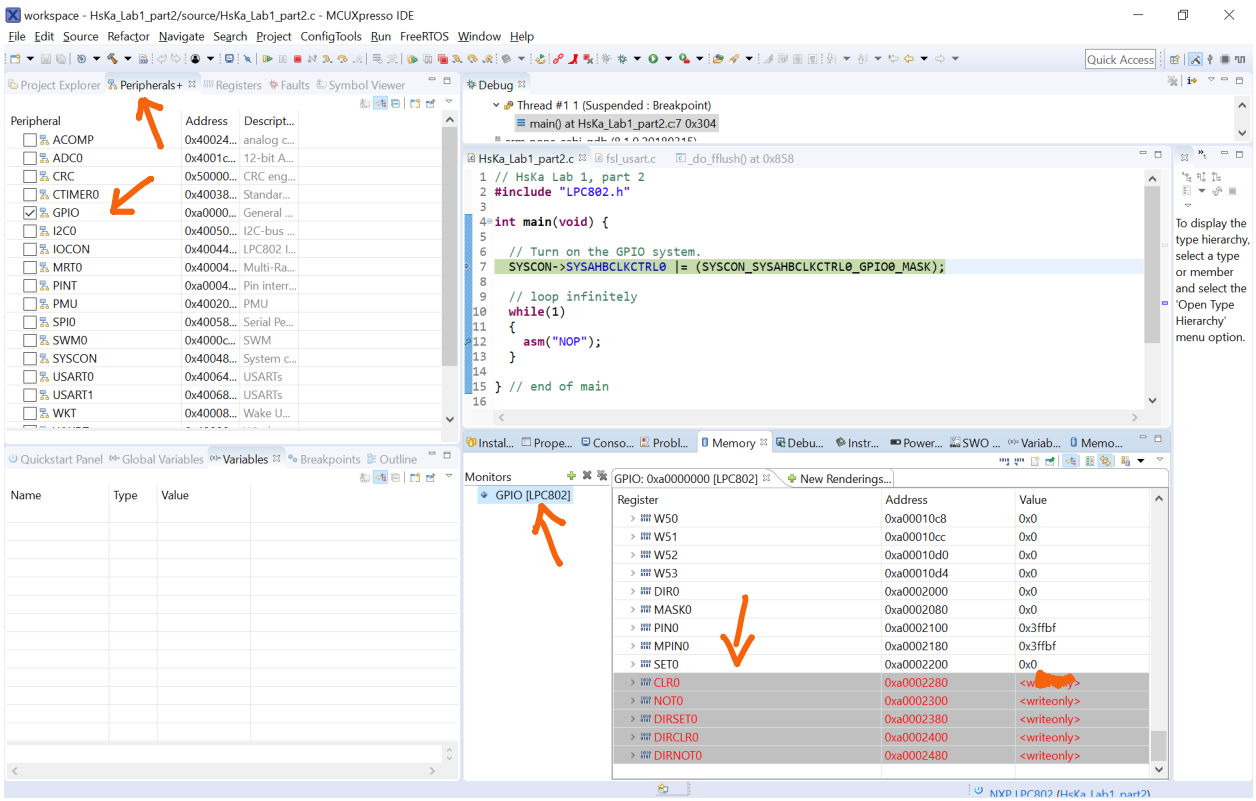


Figure 7 The Peripheral panel has been selected in the upper right and the GPIO peripheral selected with the checkbox. The results in the GPIO peripheral registers being added to the Memory panel in the bottom of the IDE. Scroll down to get the write-only registers, in red.

Change LED State using Peripheral Register View

The first thing we will do is change the state of the Red LD5 LED.

From the schematic diagram for the board we see that this LED is connected to the LPC802's GPIO 9 pin (physical pin 13 on the actual chip).

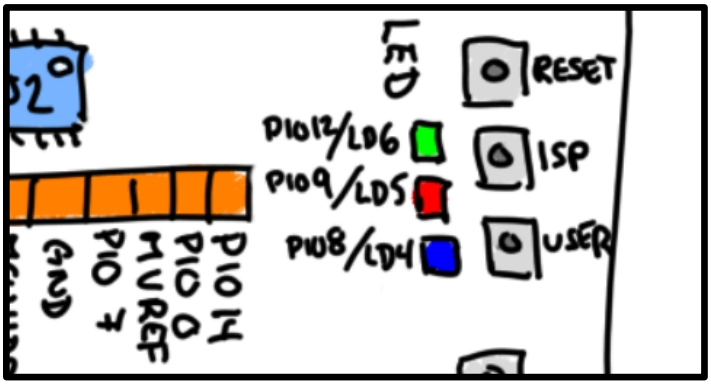


Figure 8 Closeup view of the buttons and LEDs for the board. GPIO 8 can be either an LED or the user button.

LPC 804	On the LPC804 we will use GPIO 12, which connects to the green LED ("D2") on the OM40001 board.

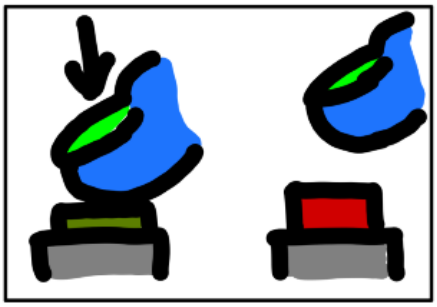
The LPC802 User Manual shows us that to make the voltage change on GPIO 9 pin we need to modify the Direction set (DIRSET), set (SET) and clear (CLR) registers, also known as DIRSET0, SET0 and CLR0 (or DIRSET[0], SET[0] or CLR[0], depending on context):

1. Activate the GPIO **in your C code**:
    a. From the C code, use your debugger to step through the line in your source code that sets the GPIO0 (Bit 6) bit of SYSAHBCLKCTRL0 register in the SYSCON
        i. Use the C Code: `SYSCON->SYSAHBCLKCTRL0 |= SYSCON_SYSAHBCLKCTRL0_GPIO0_MASK;`
2. With your code paused inside the while loop, on the asm("NOP"); line, set the direction of the pin **using the Peripherals panel**:
    a. Set Bit 9 of the DIRSET[0] register in the GPIO
        i. In the DIRSET0 register insert the value 0x200 in hexadecimal (this is equivalent to 0b1000000000)
3. Still on the assembler line, turn on the LED (it may already be on) **using the Peripherals panel**
    a. Set Bit 9 of the CLR[0] register in the GPIO
        i. In the CLR0 register insert the value 0x200
4. Still on the assembler line, turn off the LED **using the Peripherals panel**
    a. Set Bit 9 of the SET[0] register in the GPIO
        i. In the SET0 register insert the value 0x200

LPC 804	For the '804 activate the Green LED, D3, connected to **PIO0_12.** Therefore, in Steps 2, 3 and 4: don't use Bit 9. **Use Bit 12** because it's GPIO 12 on the '804. That would require a hexadecimal value of **0x1000** if you want to set Bit 12 to 1.

We will ignore the other buttons and LEDs for now.

Read a Button State Using Peripheral Register View



Our second task is to **use the debugger and the Peripherals view** to assess the state of the button. Run the code in Figure 7 (or at the beginning of this section) and halt it within the for loop. Note that just before the for loop you have activated the GPIO module, so pins associated with GPIO are active. By default, all GPIO pins are configured to be inputs.

Activate the Peripheral view, as per Figure 7 and make sure that GPIO is selected. At the bottom of the MCUXpresso window, scroll down to the write-only registers in red. Enter the following information into the following register:

- Set GPIO 0 **Bit 8** to input:
  - make **bit 8** of the DIR**CLR**0 a 1.
  - In binary, that's a **nine-bit** binary sequence: 0b100000000 (zero-b-one-and-eight-zeroes)
  - We need to put a hexadecimal equivalent into the DIR**CLR**0 register: 0x100.

Now, test the switch. Make sure to have a breakpoint on the asm("NOP"); line. Hit the "Resume" button (or the "F8" key on your keyboard) and the program will proceed and loop once.

The input value for GPIO 8 will be stored in Byte Register 8 (B8).[4] Scroll up from the write-only registers. What is the value of Byte Register 8? Now, press the button and hold it. Hit the Continue button again. Did the value of Byte Register 8 change? It should have changed colour and the value of Byte Register 8 should have changed with it.

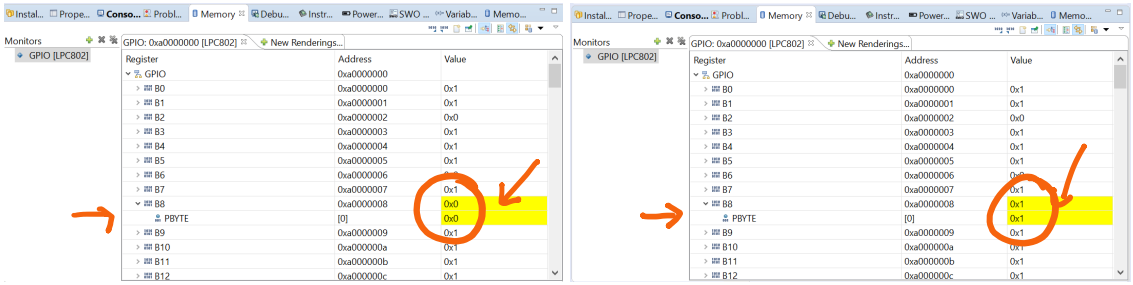


Figure 9 When the button is pressed down the B8 byte register gets a 0 value. When the button is released it becomes 0x1. Each time there is a change the register shows this by turning yellow

All students: demonstrate to the lab teaching assistant (TA) that you can turn the LED on and off by alternatively setting bit 9 in GPIO's CLR0 and SET0 registers. Also, show the instructor that the value in GPIO's B8 Byte Registers updates when you press the User button. Updating means that it will alternate between 0 and 1 when you press the User button.

LPC 804	For the '804 we will use the "user button" that is connected to PIO0_13 (that's GPIO 13). You'll need to put a value of 0x2000 into the DIRCLR0 register. To check on the state of the switch look at Byte Register 13 (B0-13).

[4] In some versions of the IDE the Byte Register is B0- followed by a particular number. For instance, Byte Register 8 would be B0-8 and Byte Register 13 would be B0-13.

Part 3: A C Program to Turn on an LED

Background material

You can access all of the registers and bits in a C program. The best way to see this is through an example. Here, modify LED LD5 (connected to GPIO 9 on the LPC802 board). Write the following program to do so:

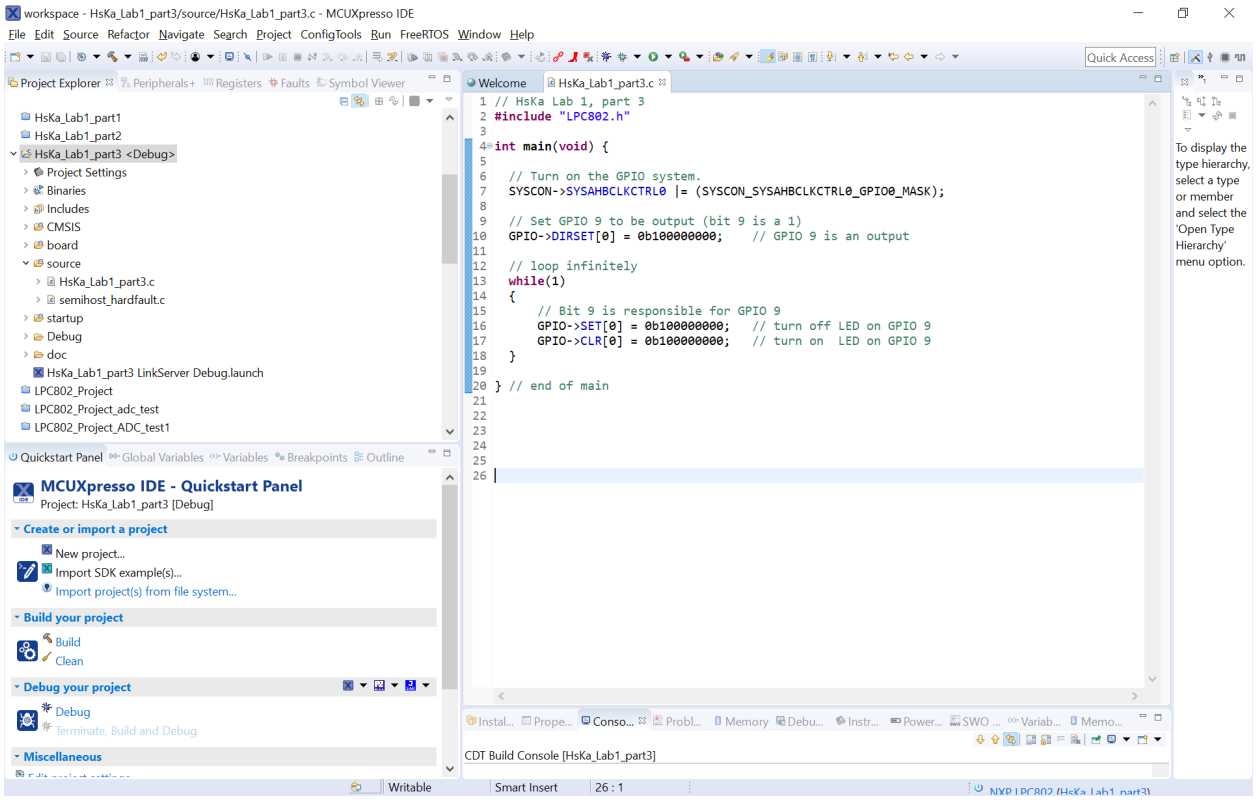


Figure 10 A simple LED blinking program for LED LD5 on the microcontroller's GPIO 9 pin.

Compile the code, then download the code onto your board using the blue debug "bug" in the IDE. It will connect to your board and send the executable code. Then it will pause at the SYSCON line. Use the "Step Over" button (F6 on your keyboard) to step over the Direction Set line and then to enter the While loop where the GPIO output on Pin 9 is set and cleared.

Watch the LED as you step through your program (keep hitting the F6 key on your keyboard).

LPC 804	For the '804 activate the Green LED, D3, connected to **PIO0_12.** For the binary sequence for the direction set, set and clear registers, **use Bit 12** because it's GPIO 12 on the '804. That would require a binary value of **0b100000000000** (or a hexadecimal value of 0x1000).

Part 4: Wrap-up

Reflection on Learning

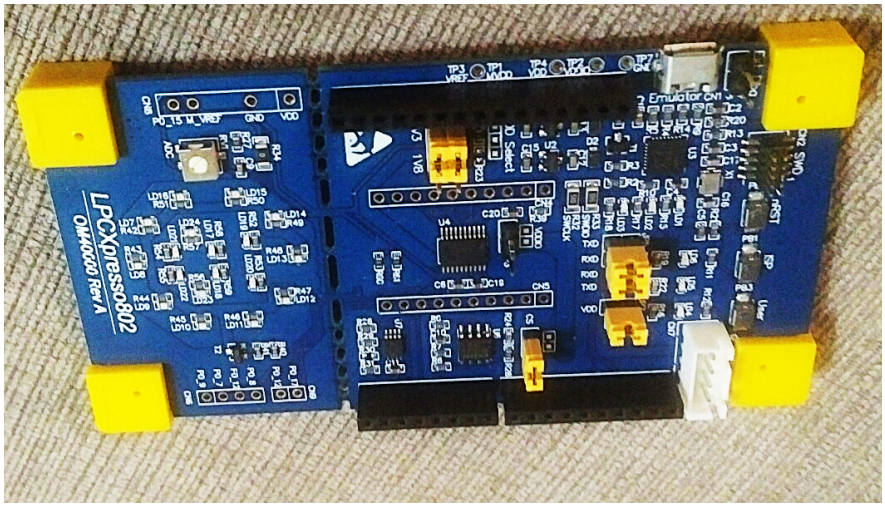
Talk to the other students in the lab. Can you see how using the debugger features allows you to better understand the inner workings of the microprocessor? Are there other ways to view memory on the chip that could be useful? If you were to start an engineering project with a microcontroller, how much would it cost you to get a stand-alone debugger like a PICKit4 or a Segger J-Link (EDU or standard)? Compare that to the standard hourly wage for an Electrical, Software or Mechatronics engineer. Does it sound like it would be worth it to use one in future projects?

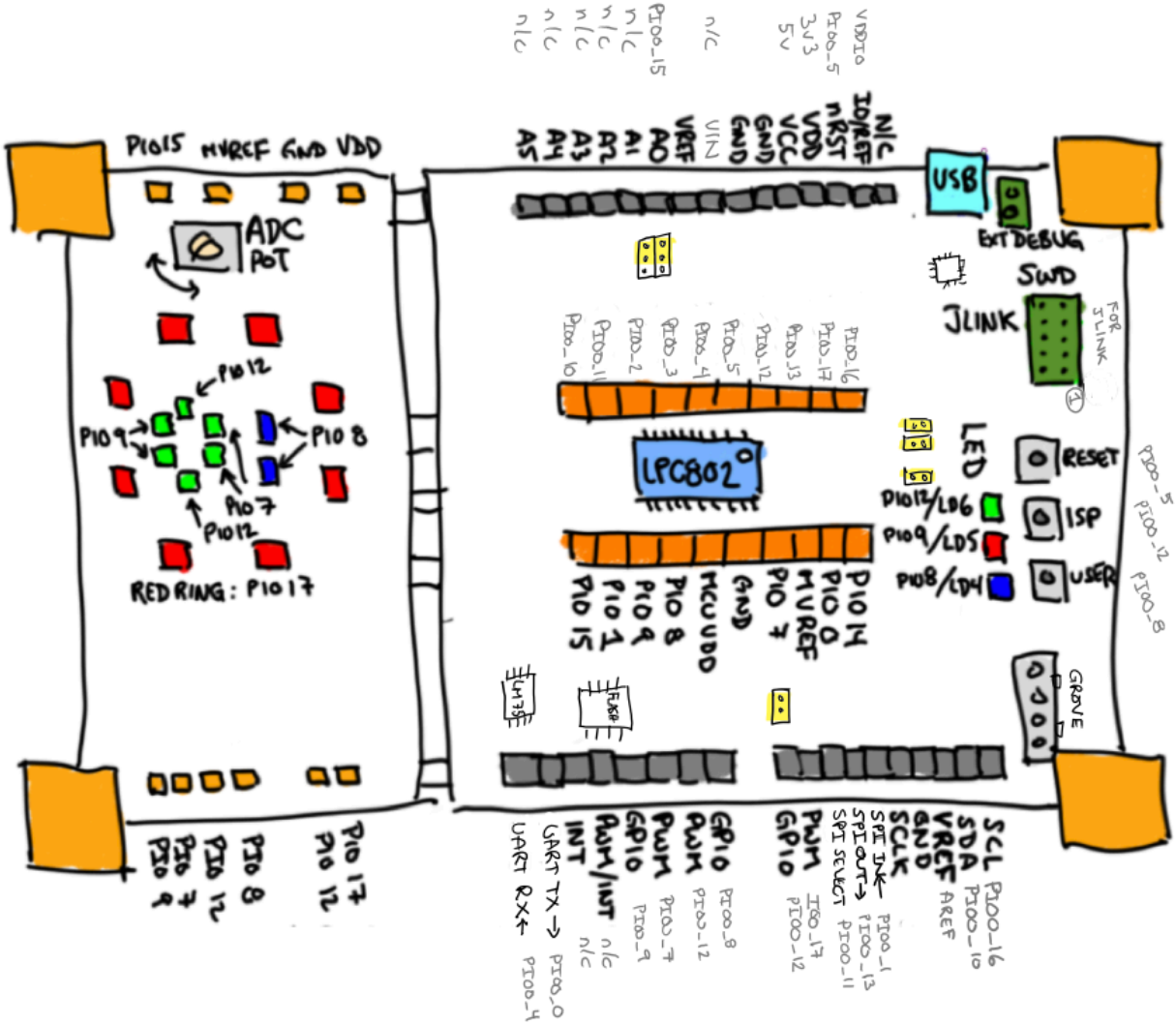
Communication – Reporting

There is no report for this lab. Simply ensure that you perform the demonstrations during the lab session.

Appendix

The LPC802 board (OM40000) and pinout description





The LPC804 board (OM40001) and pinout description