Week 4 Lab

CC2511

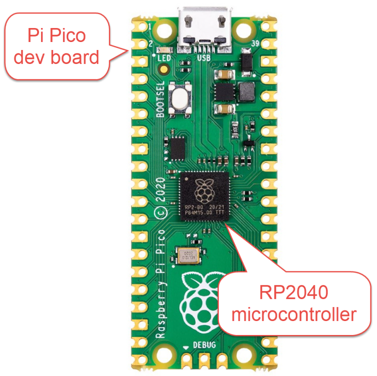
This week, you will write a microcontroller program **without using the SDK**. Your objective is to control a LED on the development board. You'll also learn how to read a microcontroller datasheet to locate the registers for a specific task.

Task Description

The Raspberry Pi Pico board has a single on-board LED, which can be switched on or off under program control. Your task is to demonstrate the use of the LED by writing a program that repeats the following steps: switch it on, wait a set time, switch it off, wait a set time, repeat. **Follow the steps below to learn how to configure the microcontroller to drive the LED**.

Documents

There are two essential documents when working with the Pico development board. You should make sure that you have an electronic copy of each to hand whenever you work with the development boards. You can download all necessary documents from LearnJCU.



- **RP2040 Datasheet.** This contains a full definition of the RP2040 microcontroller, including programming information.
- **Raspberry Pi Pico Datasheet.** This describes the Pi Pico development board module, including a schematic.

Terminology

- A **pin** is a connection between the microcontroller and the rest of the circuit.
- The pins on the RP2040 are divided into two **banks**, the QSPI bank and the User bank. The 30 User bank pins are name *GPIO0* to *GPIO29*.
- Hardware **registers** are reserved addresses in memory that allow the programmer to control and detect aspects of the microcontroller's behaviour. For example, by writing to a specific register, you can set a general purpose input-output pin's voltage to be high or low. By reading a specific register, you can determine the latest character received via a serial port.

Examining the hardware

Identifying the microcontroller pins

control the pin -> control the LED

The Pico boards have a single on-board green LED connected to a digital output pin on the microprocessor. To use this LED, follow these steps:

QUESTION—To power on the LED, do you set the output pin to logic HIGH or logic LOW?

1. Open up the Pico Datasheet and locate the schematic for the Pico board.
2. Locate the LED on the schematic.
3. By following the schematic, **determine which microprocessor pin is connected to** the LED. Complete the following table[1]:

LED colour	Microcontroller pin
Green	GPIO25

[1] You do not need to write the full name of the pin. Here, you can use the GPIO name which is of the format "GPIOxx", e.g. GPIO1, GPIO29, etc.

Pad Control

'Each GPIO is connected to the off-chip world via a "pad". Pads are the electrical interface between the chip's internal logic and external circuitry.'[2] You need to ensure that the pad for each of the GPIO pins you will be using is enabled. You can find the address of each register in *Table 338* of the **RP2040 Datasheet** (in section 2.19.6.3. *Pad Control - User Bank*).

[2] **RP2040 Datasheet** *Section 2.19.4*

You will need to determine the address of each register. You can do this by first finding the base address for this bank of registers, and then finding the offset for each specific register. Note that all these addresses and offsets are in hexadecimal format ('hex').

Microcontroller pin	Base address	Offset
GPIO25	0x4001c000	0x68

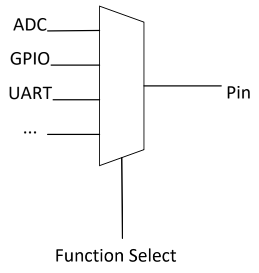
Now look at the register description for the register. (Hint: click on the appropriate link in the table.)

In order to make sure that the pad functions properly, you will need to enable both pad input and pad output. Thus you must set *Output disable* to off (0) and *Input enable* to on (1).

Flag	Bits	Value
Output disable	7	0
Input enable	6	1

Signal Multiplexing

The microprocessor has many different blocks of functionality, such as different communication interfaces, voltage sensing, and digital input/output. Each physical pin is routed through a multiplexer that specifies which of these signals is the one to be connected through to the pin. Our device describes these as "functions".



In the **RP2040 Datasheet** *Chapter 2.19* there is a table (*Table 278*) listing the Function Select modes which can be set for each pin. The "Function" for each pin defines which signal is connected to each pin.

We will drive the LEDs by using the microprocessor's Single-Cycle input/output (*SIO*) function. In SIO mode, each pin can be configured to generate either a low voltage (0 V) or a high voltage (3.3 V). To use a pin as general purpose input/output (GPIO), the relevant "*Fx*" function identifier must be selected[3]. **You must identify which Function identifier is needed to use the pin for SIO.** Look for the value SIO in the row containing each relevant GPIO pin, and note which column contains "SIO".

[3] Examples of Fx function identifiers are F1, F2, F3, ...

Complete the following table for the pin that you identified above:

Microcontroller pin	Function to enable GPIO
GPIO25	F5

GPIO Control

The function allocated to each GPIO is selected by writing to the FUNCSEL field in the GPIO's CTRL register. Locate the documentation for the relevant register.

HINT—the CTRL register will be named GPIOxx_CTRL (e.g. GPIO18_CTRL for GPIO18). Use the Find facility (Ctrl+F) in the Datasheet to find the appropriate section of the documentation.

You will need to determine the address of each register. You can do this by first finding the base address for this bank of registers, and then finding the offset for each specific register. Note that all these addresses and offsets are in hexadecimal format ('hex').

Microcontroller pin	Base address	Offset
GPIO25_CTRL ?	0x40014000	0x0cc

Now look at the documentation for the register. This describes which bits of the 32-bit register are reserved for each purpose.

You will need to locate the bits that control function selection. In you code you will need to set the value in these bits to the Function selector determined in Signal Multiplexing earlier[4]. You will also need to decide what to do with the other parts of the register. (Hint: you can safely set all the other values to 0. Look at the descriptions to see why.)

[4] Note that the function control maps to a number, so if you are using F3, the bits should be set to the value 3, and if you are using F5, they should be set to 5, etc.

Data Direction

SIO pins can be configured as either digital input or digital output. In digital input mode, the microprocessor senses whether the voltage on the pin is high or low. In digital output mode, the microprocessor acts as a voltage source and generates either a low voltage or a high voltage. This setting is referred to as the data direction.

In the **RP2040 Datasheet**, examine *Section 2.3.1.7 List of registers*. Observe that there are several registers for controlling whether GPIO outputs are enabled, including GPIO_OE, GPIO_OE_SET, GPIO_OE_CLR and GPIO_OE_XOR. We are going to use GPIO_OE_SET. See 'Controlling each LED' below for a discussion of why this is the best variant to use.

Fill in the table below with the address of the register used to configure the GPIO pins as outputs.

Register	**Base address**	**Offset**
GPIO_OE_SET	0xd0000000	0x024

Controlling each LED

A GPIO pin can output either logic low (aka logic 0, which is 0 volts), or logic high (aka logic 1, which is 3.3 volts on this system). There are multiple ways to interact with the GPIO pins.

1. You can set the value directly by writing into the **GPIO output value** register (GPIO_OUT). There is a single register for the entire set of pins, and each bit corresponds to a pin. The least significant bit corresponds to pin 0. For example, the binary value $\overset{7654}{1000}\ \overset{3210}{0111}$ would set a logic high on pins 7, 2, 1, and 0. All other pins would be zero to logic low. (Look at the documentation for register GPIO_OUT in *Table 20* of the **RP2040 datasheet**: the top two bits, 30 and 31, are decribed as *Reserved*, i.e. not used. This is because a register contains 32 bits and there are only 30 GPIO pins.)

   This mode enables you to change the value of all pins in the port in a single statement. However, if you are using multiple pins for different purposes, you need to be careful to change only the bits of interest. This can be done using bitwise logic statements such as AND or OR, e.g.

   ```
   GPIO_OUT = GPIO_OUT | (1 << 4);
   ```

   would OR the existing value with 1 shifted left 4 bits (i.e. 10000 binary) and thereby turn pin 4 on. Similarly, you can use AND NOT to turn off a bit:

EXTENSION TOPIC—The code here (writing into GPIO_OUT) has an issue. Can you see it?
The CPU must read the value of the register, then compute the OR, then write it back. If the register should change in the meantime (perhaps because this code was interrupted by another task), the *old* value of the other bits will be restored. This is called a *race condition* and can cause intermittent bugs. The value clear and value set registers are preferred in this case.

$\rightarrow$ GPIO_OUT = ITSELF OR 00010000, setting the 5th bit to 1

NOT operator

the 4th bit

```
GPIO_OUT = GPIO_OUT & ~(1 << 4);
```

A AND A' = 0, therefore switching it to 0 every time

would turn pin 4 off.

2. An alternative interface allows direct control over each pin without interfering with the others. This can be achieved using the **GPIO output value set register** (GPIO_OUT_SET) and the **GPIO output value clear register** (GPIO_OUT_CLR). Each binary 1 written into the clear register will clear the corresponding pin (to zero); whereas each binary 1 written into the set register will set the corresponding pin to one. For example, the binary value 1000 0000 written into the clear register would cause pin 7 to be changed to zero whereas all other pins would be unchanged. A convenient way to generate this binary value would be `(1 << 7)`.

Register	Base address	Offset
GPIO_OUT_SET	0xd0000000	0x014
GPIO_OUT_CLR	0xd0000000	0x018

Your task is to control a single pin: you determined which one earlier by referring to the Pico schematic. GPIO25

Action	Register	Value
Switch on	GPIO_OUT_SET	0x1 << 25
Switch off	GPIO_OUT_CLR	0x1 << 25

Writing the software

Creating a Pico Project

To create the new project, open a Developer Command Prompt for VS 2022. Navigate to the folder containing your main git repository, and type the following at the command line:

```
python %PICO_SDK_PATH%\..\ppgen\ppgen.py Lab4
```

If the generator is not present on your computer, follow the instructions in the Week 3 lab: **Install or update the project generator**.

This will create a new folder, Lab4, and populate it with source files and project files. Open the project in VS Code by typing the following:

```
code Lab4
```

This instructs the VS Code editor to start, and to open the new folder. As the project opens, you will be asked to select a kit: select **GCC *N.N.N* for arm-none-eabi**. (N.N.N will depend on which version of the GCC tool-chain is installed.)

Open the project

Now open the project, by typing the following at the same command line:

```
code Lab4
```

This instructs the VS Code editor to start, and to open the new folder.

Inspect the code

Open main.c, and you will find the barest of programs:

```c
#include "pico/stdlib.h"
#include <stdbool.h>
int main(void) {
  // TODO - Initialise components and variables
  while (true) {
    // TODO - Repeated code here
  }
}
```

Delete, this is lab 2 on hard mode

Normally, the stdlib.h file from the Raspberry Pi Pico SDK is a very useful starting point: it contains helper functions and structures for many common operations. However, for this project you should learn how to do everything for yourself when working with the hardware; so this time – delete the top line!

The next line includes the standard file stdbool.h. A boolean type bool and values true and false are not guaranteed to be part of the C language; this file provides them, and makes for more readable code, for example in the while statement.

OBSERVATION—Normally you will use the SDK whenever possible, because it is more efficient, and reduces the likelihood of errors. However, there will always be times when a particular functionality has not been included in the SDK adnd you will need to work out the code for yourself.

The program has one function, named main(), which will automatically be executed when the program starts. Within the function there is a place to put initialisation code which will run only once and a place for the repeated code that will run forever.

For this lab, the initialisation code will need to include code to:

1. enable the pads;
2. select the GPIO function; and
3. set the data direction.

The repeated code will comprise the code to turn the LED on and off, with a timed waiting period between each action. This will run forever, so it is placed inside a **while**(true) {...} loop.

Addresses & other constants

The addresses that you looked up earlier should be added to the source code as constants. A standard way to define constants in 'C' is to use macros. For example, you could define the SIO base address and the GPIO_OE_SET register as shown below.

Observation—Notice that - unusually for 'C' - there are no semi-colons after macros.

```
#define SIO_BASE         0xd0000000
#define REG_GPIO_OE_SET  (SIO_BASE+0x024)
```

It's also a good idea to add any GPIO pin numbers in your program, so that you can use names in code, instead of numbers.

```
#define LED_PIN          29 // Note:- 29 is NOT the correct number
```

Types

All the registers used in this lab are 32 bits wide, so we need to use a data type that is a 32-bit wide unsigned integer. Again, normally this would be defined for you in a helper file, but today you should add it yourself, as follows:

```
typedef unsigned long uint32_t;
```

Reading and writing registers

When you read a 32-bit register, or when you write to it, you need to treat it as an unsigned 32-bit number, and thus you need to convert its address to a pointer to an unsigned 32-bit number. A macro like CONTENT_OF below can be very helpful here:

```
#define CONTENT_OF(addr) (*(volatile uint32_t*)addr)
```

You could use it to set the GPIO_OE_SET register like this:

```
uint32_t some_value = 1;
CONTENT_OF(REG_GPIO_OE_SET) = some_value;
```

And you could use it to change just one bit of a register value, like this:

```
CONTENT_OF(REG_GPIO_OE) = CONTENT_OF(REG_GPIO_OE) | 0x02;
```

Doing nothing

The Pico SDK contains helper functions to let your program wait idly for a while, but today you can write your own wait loop, like this:

```
for (uint32_t i = 0; i < 8000000; i++) {
  __asm volatile ("nop");
}
```

Extension—The keyword **volatile** is used to prevent the standard compiler behaviour: noticing that the code does nothing and removing it. This is often referred to as 'optimising it out'.

Note that this code directly invokes some assembly code, using the **__asm** keyword. We'll be looking at integrating assembly code and 'C' code in depth later. For now, just take on board that we can go down to an even lower level than 'C' whenever we need to.

Build your project (press the **F7** key), and make sure that there are no build errors.

Testing the program

Deliver the program

Deliver your initial program to the Pico board the same way as in the last lab:

1. Remove the USB cable from the Pico board.
2. Holding the BOOTSEL button down, reconnect the USB cable.
3. Release the BOOTSEL button.
4. A new Windows Explorer window will open, displaying a virtual drive.
5. In your project's build subdirectory, locate the file Lab4.uf2 and copy it the virtual drive.

The program will be delivered to your Pico board, which will immediately run the program.

Your task

1. Write code to initialise the LED pin and then to cycle through the on/off sequence. You might find it helpful to write a function that waits for a while to reduce repetition in your code.
2. Deliver the code to your Pico board.

Solve simpler problems first

Some suggestions to help you build up to the task requirements:

- As a first step, just write a program that switches on the LED. Then you will know whether your initialisation code is working correctly.
- If you can't see the LED flash on and off, change the wait time.

Assessment

To finish this lab, you must:

- Demonstrate a working board that continuously switches the on-board LED on and off.
- Show your prac tutor your GitHub website including your Lab 4 code (see the instructions for GitHub in last week's lab if necessary).