Week 5 Lab

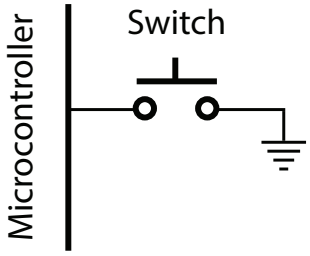
CC2511

Firstly, you will read digital input from a floating pin to learn to recognise this common error; and secondly, you will use pulse width modulation to smoothly adjust the LED colours.

Part 1—Floating pins / unconnected inputs

A "floating" pin is an input that is unconnected. This could happen because of a broken circuit, a bad solder joint, etc.

To help you recognise the symptoms in your own designs, you'll deliberately create a floating pin.



A simple switch circuit with a floating pin. How would you modify the circuit to avoid the problem? Sketch a new version.

Getting started

This week, use the same breadboard setup as in Lab 3 or, if it is ready, use your CC2511 Dev Board.

To create your project, open a Developer Command Prompt for VS 2022 and use the following command, which pulls the PWM libraries into your project:

```
python %PICO_SDK_PATH%\..\ppgen\ppgen.py Lab5 -l pwm
```

Add USB redirection for the serial port as in Lab 3, by adding `pico_stdio_usb` to the line in CMakeLists.txt that contains `target_link_libraries(...)`.

Add the code below to make use of these macro definitions, similar to those first seen in Lab 3:

```
#define CONTENT_OF(addr)            (*(volatile uint32_t*)addr)
#define PAD_CONTROL_BASE            0x4001c000
#define REG_PAD_CONTROL_GPIO15      (PAD_CONTROL_BASE+0x40)
#define CTRL_PDE                    0x4
#define CTRL_PUE                    0x8
```

Write the code

Bits	Name	Description	Type	Reset
3	PUE	Pull up enable	RW	0x0
2	PDE	Pull down enable	RW	0x1

1. Select a pin that is not connected to anything other than the header around the outside of the board. Use GPIO15.
2. Search the RP2040 Datasheet to learn how to configure the internal pullup and pulldown resistors. Hint: look in *2.19.6.3. Pad Control - User Bank*.
3. During initialisation, add code to clear the PUE (Pull up enable) and PDE (Pull down enable) bits of the appropriate register.

Remember that you can use `|` to turn a bit on and `& ~` to turn it off, e.g.:

```
X &= ~0x8; // Turns off bit 3 in X
X |= 0x8; // Turns on bit 3 in X
```

4. Create an endless loop that reads the digital input from your chosen pin and displays the result using the **blue LED**. If the pin reads as high, then turn the blue LED on, and if it reads as low, then turn the LED off.

5. Connect a short wire into the header connected to your pin and touch it with your finger. You should see the LED flicker unpredictably as you touch the wire. You might need to experiment with this; floating inputs are inherently unpredictable. Sometimes just waving your hand **near** the wire is enough.

Avoid the floating input

Some microcontrollers (including ours) include optional internal pullup or pulldown resistors. A pullup resistor is a large resistance (e.g. > 10 kΩ) connected to Vcc, so that the pin is *weakly* pulled high. Similarly, a pulldown resistor connects to ground so that the pin is weakly pulled low.

1. Modify your code so sending a character over the serial port will change the state of the pullup/pulldown resistors. This enables you to switch it on or off in software and experiment with its influence on the input pin.

   - If you receive 'u' turn the pullup on and the pulldown off.
   - If you receive 'd' turn the pullup off and the pulldown on.
   - If you receive 'o' turn both pullup and pulldown off.
   - In each case send an appropriate message to the terminal, e.g. "Pin 15 to pullup", "Pin 15 to pulldown" or "Pin 15 to pull none".

2. Verify that the presence of the pullup or pulldown resistor avoids the unpredictable behaviour that you observed earlier.

Conclusion

Hopefully you will recognise the symptoms of a floating pin if you should ever see one. It could save you quite some time in debugging a broken circuit.

Part 2—Pulse Width Modulation

Next, extend your program so that the other two colours (red and green) have variable light intensity. Different intensities can be generated using pulse width modulation (PWM). If the period is fast enough, the flickering will be invisible to the eye, and it will appear as though the LED brightness is changing.

Design a serial port interface to allow interactive adjustment of the RGB colours. For example, pressing "r" might increase the amount of red whereas "R" might decrease the amount of red.

Look at the Week 5 Lecture 2 notes for help on initialising and controlling PWM.

- You will need to set the GPIO function for each pin.
- You will need to enable PWM for each pin.
- You will need to set the PWM level for the pin whenever the matching keys are pressed.

Hints

- Use an integer whose value ranges between 0 and 255 to store the current value of each colour's brightness.
- When you set the LED intensity use the square of the value (i.e. `red_value * red_value`). This gives a better distribution. what

Assessment

To finish this lab, you must demonstrate to your tutor:

- The source code in Part 1 where you enable the pullup or pull-down resistor.
- Smoothly varying colours on the LED that are controlled over the UART. Each of the two channels must be individually controllable.
- Show your prac tutor your GitHub webpage where your source code is uploaded.