A white background with black dots

Description automatically generated

**LABORATORY MANUAL**

**GPIO**

**AY23/24**

**OBJECTIVES**

* 1. Get to know the Raspberry Pico W
  2. Learn how to use registers for configuration
  3. Get to understand and use library functions
  4. Configuring GPIO as input and output

**EQUIPMENT**

1. Computer or laptop that has the Pico Build tools installed
2. Raspberry Pico W
3. Micro-USB Cable

**NOTE:**

1. Only students wearing fully covered shoes are allowed in the lab due to safety.
2. Bring your laptop with Code Composer Studio installed.
3. For your understanding and a better quiz preparation, take notes about the given tasks, especially questions or unexpected code behaviour.

**Introduction**

In this lab session, we learn about the development platform Raspberry Pico W and the Pico C/C++ SDK, which are used throughout all ES labs. We will also familiarise you with concepts like direct register access, polling, and serial communication. The following introduction should give a broad overview of the working environment, but it is unnecessary to comprehend all the details to complete this laboratory.

**RP2040 and Raspberry Pico W**

RP2040 is the debut microcontroller from Raspberry Pi. It brings our signature values of high performance, low cost, and ease of use to the microcontroller space.

With a large on-chip memory, symmetric dual-core processor complex, deterministic bus fabric, and rich peripheral set augmented with our unique Programmable I/O (PIO) subsystem, it provides professional users with unrivalled power and flexibility. With detailed documentation, a polished MicroPython port, and a UF2 bootloader in ROM, it has the lowest possible barrier to entry for beginner and hobbyist users.

RP2040 is a stateless device, with support for cached execute-in-place from external QSPI memory. This design decision allows you to choose the appropriate density of non-volatile storage for your application, and to benefit from the low pricing of commodity Flash parts.

RP2040 is manufactured on a modern 40nm process node, delivering high performance, low dynamic power consumption, and low leakage, with a variety of low-power modes to support extended-duration operation on battery power

**Key features:**

1. Dual ARM Cortex-M0+ @ 133MHz
2. 264kB on-chip SRAM in six independent banks
3. Support for up to 16MB of off-chip Flash memory via dedicated QSPI bus
4. DMA controller
5. Fully-connected AHB crossbar
6. Interpolator and integer divider peripherals
7. On-chip programmable LDO to generate core voltage
8. 2 on-chip PLLs to generate USB and core clocks
9. 30 GPIO pins, 4 of which can be used as analogue inputs
10. Peripherals
    1. 2 UARTs
    2. 2 SPI controllers
    3. 2 I2C controllers
    4. 16 PWM channels
    5. USB 1.1 controller and PHY, with host and device support
    6. 8 PIO state machines

Raspberry Pi Pico W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW43439 while retaining the Pico form factor. The on-board 2.4GHz wireless interface has the following features:

1. Wireless (802.11n), single-band (2.4 GHz)
2. WPA3
3. Soft access point supporting up to four clients
4. Bluetooth 5.2
5. Support for Bluetooth LE Central and Peripheral roles
6. Support for Bluetooth Classic

The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The wireless interface is connected via SPI to the RP2040 microcontroller.

Due to pin limitations, some of the wireless interface pins are shared. The CLK is shared with VSYS monitor, so only when there isn’t an SPI transaction in progress can VSYS be read via the ADC. The Infineon CYW43439 DIN/DOUT and IRQ all share one pin on the RP2040. Only when an SPI transaction isn’t in progress is it suitable to check for IRQs. The interface typically runs at 33MHz.

For best wireless performance, the antenna should be in free space. For instance, putting metal under or close by the antenna can reduce its performance both in terms of gain and bandwidth. Adding grounded metal to the sides of the antenna can improve the antenna’s bandwidth.

**Embedded System Documentation**

The most important documents of an embedded system are, among others, **datasheets**, **user guides**, **technical reference manuals**, **application notes**, **errata** or **schematics**. These documents are available for:

* the microprocessor at the manufacturer’s website, e. g. Raspberry (https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html#welcome-to-rp2040)
* components of the microprocessor, e. g. ARM microprocessors documentation
* hardware boards, e. g. the Raspberry Pico W
* software, e.g. the Pico C/C++ SDK
* complementary information in wikis or online forum

Therefore, every embedded system comes with many documentation files. For this lab and subsequent labs, all the necessary files are provided on the course xsite website. It is essential to have access to all parts of the documentation to use the functionality of an embedded system to its fullest extent. The documentation is typically needed to control:

* the pins peripherals are connected to
* register addresses and memory-mapped peripherals
* configuration values for modules
* bitmasks and their meaning
* the functionality of modules (e.g. serial communication modules, etc.)

**Bare-Metal Programming**

Programming bare-metal means to program an embedded system without using an underlying Operating System (OS) like Linux, FreeRTOS or a scheduler of any sort. Typically, they consist of a base loop, and all activity is either polled or interrupt-driven. It may include vendor-supplied header files or driver subroutines, but the code determines what happens when and is in total control. These systems are the most deterministic, are easier to test and debug, and are the way to go for most microcontroller-based applications that tend to be specific.

**Watchdog Timer**

The watchdog timer is an internal timer unit of an embedded system that will reset the system if it times out. Under normal operation, the system will reset the watchdog timer regularly to prevent it from timing out, but in case of an error, the watchdog timer will reset the device. This is used to avoid a deployed embedded system that cannot be easily accessed (e.g. Mars Rover) from being stuck if an error occurs by resetting it once the timer expires. Do note that the watchdog timer for the Pico is **turned off by default**. As such, if any error occurs during the execution of the instruction, the program will cease and the device will not reset by itself. To enable the watchdog timer, please see the following [example](https://github.com/raspberrypi/pico-examples/blob/master/watchdog/hello_watchdog/hello_watchdog.c).

**Defines, Macros and in-line Functions**

Preprocessor directives like defines and macros, and in-line functions are often used when programming embedded systems to make the code more readable and easier to maintain. Examples for a ***define***, ***macro*** and ***in-line*** function are given in are provided in the link below. The ***#define*** directive, used for ***defines*** and ***macros***, is a preprocessor directive that instructs the compiler to replace certain code parts as defined. In contrast to that, the ***in-line*** function is just a request to the compiler. This means, depending on the optimisation configuration of the compiler, function calls to ***in-line*** functions can be replaced with the function body or are treated like standard function calls. However, in contrast to ***macros***, ***in-line*** functions are subject to strict parameter checking and considered *safer*.

A more detailed explanation and their differences can be found on the following site: https://www.thegeekstuff.com/2013/04/c-macros-inline-functions/

***Library Functions vs Hard-Coded Register Access***

Using the installed ARM toolchain and build tools, we can build an .uf2 image of our program and flash the image to the microprocessor via USB. This task will teach you how to flash a functioning application using GPIO functions onto the microprocessor. These GPIO pins can be configured either as input or output. GPIO pins can also drive peripherals (i.e. LEDs, small actuators, relays) or read inputs (i.e. sensors, buttons). Furthermore, you will learn how to swap and extend the application using hard-coded register access and library functions.

All registers addresses of each of the GPIO pins can be found in the RP2040 [datasheet](https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf?_gl=1*e21eo8*_ga*MTEyMDkxNzI5MC4xNjc3NTA3NTU4*_ga_22FD70LWDS*MTY5MTkwOTI3Mi4xMS4xLjE2OTE5MDkzMTQuMC4wLjA.) (Section 2.19.6.1). It is encouraged to use the hardware API provided by the SDK to manipulate the GPIO pins atomically. More information on the low-level accessors to the registers can be found in “address\_mapped.h” found in the SDK.

Documentation of the hardware APIs can be found [here](https://www.raspberrypi.com/documentation/pico-sdk/hardware.html#hardware_gpio).

***Task 1.1: Configuring GPIO Pins as Output (LED)***

Complete the provided code to toggle the LED (PICO\_DEFAULT\_LED\_PIN) on and off every second using a busy wait (i.e. sleep()).

Hint: Refer to “[blink.c](https://github.com/raspberrypi/pico-examples/blob/master/blink/blink.c)”

***Task 1.2: Configuring GPIO Pins as Input***

In this task, we shall write an application that will switch the LED on or off based on a signal provided by an GPIO pin. A method to continuously sample certain information (i.e. register or input) is called polling. This can be used to access peripheral or internal status information or read data. GPIO pins can also be used for input.

In the example code, GP3 has been configured to toggle on and off every 100ms. Configure GP2 as an input pin and connect a jumper cable across GP2 and GP3. If a HIGH is read on GP2, toggle the LED on, else, toggle the LED off.

To avoid a floating pin, pull-up or pull-down resistors are used. A pin is called floating when the pin does not have a fixed voltage level, leading to unexpected behaviours if the pin is read. In Figure 1, the two concepts of pull-up and pull-down resistors are illustrated. A pull-up resistor will ensure the pin is on high voltage level when the button is not pressed (open circuit) and low if the button is pressed (short circuit). The pull-down resistor works vice versa.

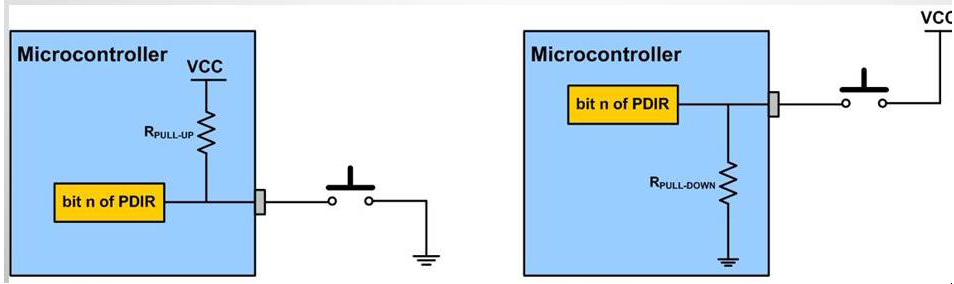


Figure 1. Schematic of a pull-up (left) and pull-down resistor

Once the GPIO is configured as an input, poll (read) for the status of GP2 and use the information to manipulate the LED.