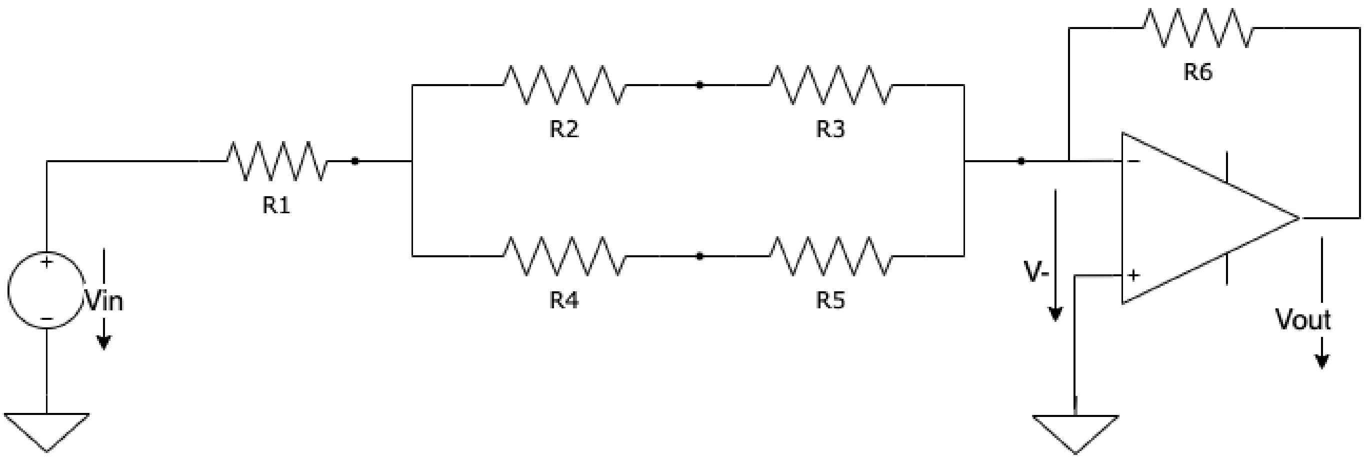
Assignment 2 2DT903

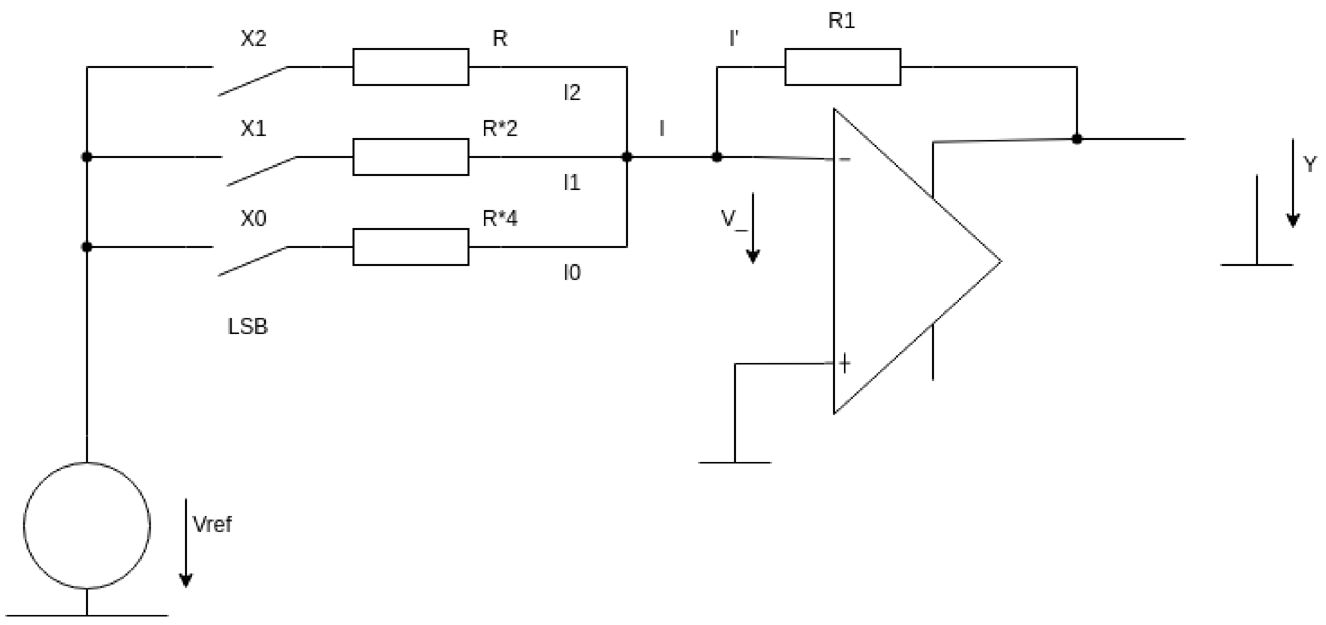
Emil Ulvagården (eu222dq@student.lnu.se)

Part 1

```
Rin = R1 + ((R2 + R4) * (R3 + R5)) / ((R2 + R4) + (R3 + R5))

Rf = R6

gclosed = -(Rf / Rin)

gclosed = -(R6 / (R1 + ((R2 + R4) * (R3 + R5)) / ((R2 + R4) + (R3 + R5))))

=> Vout = gclosed * Vin = -(R6 / (R1 + ((R2 + R4) * (R3 + R5)) / ((R2 + R4)
+ (R3 + R5)))) * Vin
```



Part 2



```
X = [x2, x1, x0]

0 <= N <= 7 # (000) <= N <= (111)

N = 4*x2 + 2*x1 + 1*x0

Vout = Vref * ((x2/2) + (x1/4) + (x0/8)) = Vref * ((4*x2 + 2*x1 + x0)/8)

MSB for x2 is 4/7

MSB for x1 is 2/7

MSB for x0 is 1/7
```

The total output volatage is a linear combination of the bits, the output voltage is proportional to the value represented by the vector X.

Part 3

Report

The project consists of two parts, one using c code and freeRTOS and the other part using microPython and a mqtt broker. Both parts of the project uses the same components but second part is connected to the a network and has the possibility to communicat with other devices while the first one can't. The first part of the project is to implement code so that when a button is pressed a led is turned on for 2 seconds and the message "Button pressed!" is printe. while that is happening another task is run where every 5 seconds a measurement of a dht11 sensor is taken and the temperature and humidity is printed unless there is an error. The second part of the project is to implement code so that a raspberry pi pico w can connect to a mqtt server and using the connection the controller can read and publish data. If a button press is detected the raspberry should send a signal to the mqtt server and when the signal is then received by the microcontroller it should turn a led on and take meassurements of the temperature and humidity with a dht11 sensor and publish the data. When the new data is published the led should turn off. There is also a timer that counts to 5 minutes and then publishes a signal to the mqtt server and the timer then restarts so that after another 5 minutes a new signal is sent.

The real-time requirement for the project is on the button press where a manual input is used and the system needs to have a fast respons. You could have a system like the project but it has more critical functions like a fire alarm or a medical alarm at a hospital where response times are important. Then RTOS is a necessaty because it needs to have a fast response time and a program written in microPython or normal c whould not be fast enough. So in systems where the response time to an event is not critical its okey to use microPython but if the system is critical RTOS should be used to insure a fast response.

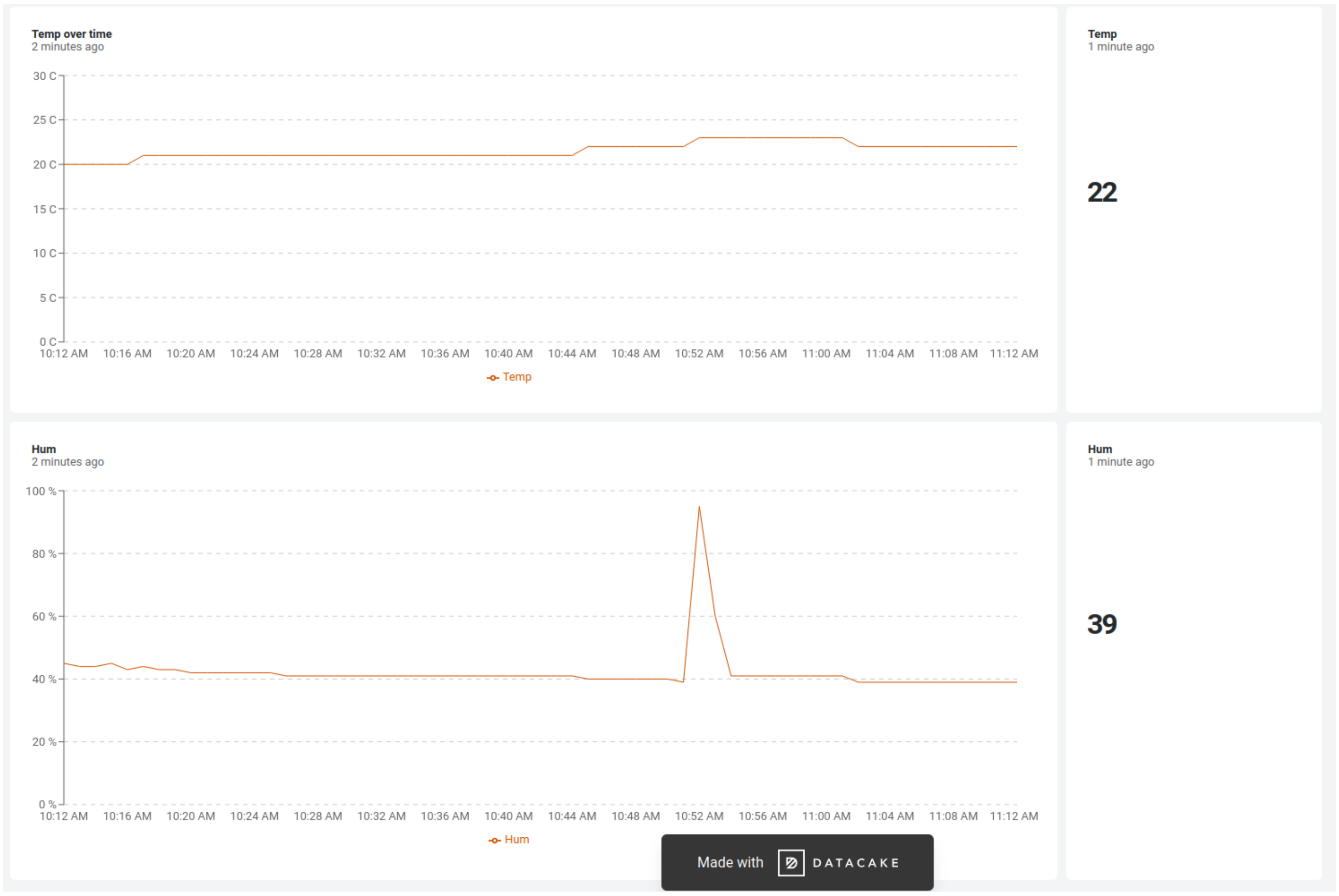
After desiding what the project should be i started by finding wiring diagrams for the different components and then connecting the different components on a breadboard. After the different components where connected i started programming the components to work one by one. When the components worked i started looking in to how to connect to a network. When connected to a network i started looking for a mqtt server that could be used and when one was found i looked for how i could connect to the server. When connected to the mqtt server i start looking at how to publish and read data on the server before implementing it with fake data to be sent. When the sent data can be read i started implementing a method for when a specific value is read the led is turned on and the dht11 sensor takes a meassurement and publishes the temperature and humidity. When the value of the dht sensor then was read from the mqtt server the led is turned off. When that part was done i went on to implement that on a button press the specifik value should be sent to the mqtt server. After that i started taking the time of when a publication was made to have regular updated data on an interval.

Picture and links

Link to the dashboard where live data is displayed on datacake

Data from dashboard on datacake

Picture of the live data from datacake with history of the latest values from the last hour in a graph. To the right there shows the latest value that has been uploaded to the mqtt server.



Save the following code on the raspberry pi pico as umqtt.py by copying the code or downloading the file from https://github.com/micropython/micropython-lib/blob/master/micropython/umqtt.simple/umqtt/simple.py and rename it to umqtt.py

Code

The js below is the code for the uplink decoder.

```
function Decoder(topic, payload) {
    try {
        // Parse the JSON payload
        payload = JSON.parse(payload);

        // Extract temperature and humidity
        var Temp = payload.t;   // Temperature
        var Hum = payload.h;    // Humidity

        return [
            {
                device: "77870f9d-679b-4a1e-aa90-b626cc8fc7ab", // Serial
Number or Device ID
                field: "TEMP",
                value: Temp
            },
            {
                device: "77870f9d-679b-4a1e-aa90-b626cc8fc7ab", // Serial
Number or Device ID
                field: "HUM",
                value: Hum
            }
        ];
    } catch (error) {
        console.error("Failed to decode payload:", error);
        return [];  // Return an empty array if decoding fails
    }
}
```

The code bellow is the code for the main program for the project in micropython.

```
import network
import time
from machine import Pin
from umqtt import MQTTClient
import ubinascii
import machine
import dht
import json  # Import JSON module for handling JSON data

# Wi-Fi credentials
ssid = ''      # Replace with your Wi-Fi network name
password = ''      # Replace with your Wi-Fi password

# MQTT server credentials
URL = "broker.emqx.io"
Port = 1883
mqtt_user = ""        # MQTT username (Empty for this broker)
mqtt_password = ""  # MQTT password (Empty for this broker)
Topic = "U/test1"
Button = Pin(0, Pin.IN)  # Set up the button on GPIO 0
LED = Pin(1, Pin.OUT)  # Set up the LED on GPIO 1
DHT = dht.DHT11(Pin(2))

# Device unique ID, make sure it's decoded as string
client_id = ubinascii.hexlify(machine.unique_id()).decode()

# Declare the client variable globally
client = None

# Function to connect to Wi-Fi
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)

    if not wlan.isconnected():
        print(f"Connecting to Wi-Fi: {ssid}")
        wlan.connect(ssid, password)
        # Wait until connection
        while not wlan.isconnected():
            time.sleep(1)
            print(".", end="")

    print("\nConnected to Wi-Fi")
    print(wlan.ifconfig())  # Print IP details

# Function to connect to MQTT server
def connect_mqtt():
    global client  # Declare client as a global variable so it's accessible
in other functions
    client = MQTTClient(client_id, URL, Port, mqtt_user, mqtt_password)

    try:
        client.connect()
        print("Connected to MQTT broker")
    except Exception as e:
        print(f"Failed to connect to MQTT broker: {e}")
        return None

    return client

# Function to publish a message to a topic
def publish_message(client, topic, message):
    try:
        client.publish(topic, message)
        print(f"Message published to {topic}: {message}")
    except Exception as e:
        print(f"Failed to publish message: {e}")

# Callback function to handle received messages
def message_callback(topic, msg):
    print(f"Received message: {msg.decode()} from topic: {topic.decode()}")

    # Turn on the LED if the message is "1"
    if msg.decode() == "1":
        LED.on()
        DHT_Sensor()  # Read sensor data and publish it
    else:
        LED.off()

# Function to subscribe to a topic
def subscribe_topic(client, topic):
    client.set_callback(message_callback)

    try:
        client.subscribe(topic)
        print(f"Subscribed to {topic}")
    except Exception as e:
        print(f"Failed to subscribe to topic: {e}")

# Debouncing function
```

```
def button_pressed():
    # Check if button is pressed and apply a short delay to debounce
    if Button.value() == 1:  # Button is active-low (pressed when 1)
        time.sleep(0.05)  # Short debounce delay of 50 ms
        if Button.value() == 1:  # Check again to confirm the button is
still pressed
            return True
    return False

# Function to read DHT sensor and publish data
def DHT_Sensor():
    DHT.measure()  # Trigger the DHT11 to take a measurement
    temp = DHT.temperature()  # Read temperature
    hum = DHT.humidity()  # Read humidity
    print("Temp:", temp, "Hum:", hum)

    # Create a JSON payload
    payload = json.dumps({"t": temp, "h": hum})  # Create JSON object
    publish_message(client, Topic, payload)  # Publish the JSON payload

# Main program
def main():
    # Connect to Wi-Fi
    connect_wifi()

    # Connect to MQTT broker
    mqtt_client = connect_mqtt()

    if mqtt_client:
        # Subscribe to a topic
        subscribe_topic(mqtt_client, Topic)

        # Initialize the last publish time
        last_publish_time = time.time()

        try:
            while True:
                mqtt_client.check_msg()  # Check for incoming messages

                # Check if the button is pressed with debouncing
                if button_pressed():
                    publish_message(mqtt_client, Topic, "1")
                    #time.sleep(0.3)  # Additional delay to prevent rapid
retriggering

                # Check if 10 seconds have passed since the last message
                current_time = time.time()
                if current_time - last_publish_time >= 300:  # 300-second
interval or 5 minutes
                    publish_message(mqtt_client, Topic, "1")
                    last_publish_time = current_time  # Reset the last
publish time

        except KeyboardInterrupt:
            print("Disconnecting from MQTT...")
            mqtt_client.disconnect()


# Run the program
if __name__ == "__main__":
    main()
```

The code bellow is for the c code in the project and the code should be saved in Project.c

```


#include <stdio.h>
#include <dht.h>
#include "pico/stdlib.h"
#include "FreeRTOS.h"
#include "task.h"

#define BUTTON_PIN 0
#define LED_PIN 1
#define DHT11_PIN 2
static const dht_model_t DHT_MODEL = DHT11;
dht_t dht;

void temp_task(void *pvParameters) {
    while (1)
    {
        dht_start_measurement(&dht);

        float temp;
        float hum;
        dht_result_t res = dht_finish_measurement_blocking(&dht, &hum,
&temp);

        if (res == DHT_RESULT_OK) {
            printf("Temperature: %.1f °C, Humidity: %.1f%%\n", temp, hum);
        } else if (res == DHT_RESULT_TIMEOUT){
            printf("DHT sensor not responding. Please check your
wiring.\n");
        } else {
            assert(result == DHT_RESULT_BAD_CHECKSUM);
            puts("Bad checksum");
        }

        vTaskDelay(pdMS_TO_TICKS(5000));
    }
}

void buttonTask(void *pvParameters) {
    bool button_pressed = false;

    while (1)
    {
        if (gpio_get(BUTTON_PIN) == 1) {
            gpio_put(LED_PIN, 1);
            printf("Button pressed!\n");
            button_pressed = true;
            vTaskDelay(pdMS_TO_TICKS(2000));
            gpio_put(LED_PIN, 0);
            button_pressed = false;
            vTaskDelay(pdMS_TO_TICKS(500));
        }
        vTaskDelay(pdMS_TO_TICKS(50));
    }
}

int main() {
    // Initialize stdio for USB or UART communication
    stdio_init_all();

    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_down(BUTTON_PIN);

    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    gpio_put(LED_PIN, 0);

    dht_init(&dht, DHT_MODEL, pio0, DHT11_PIN, true);

    xTaskCreate(buttonTask, "Button Task", 256, NULL, 1, NULL);
    xTaskCreate(temp_task, "Sensor Task", 512, NULL, 1, NULL);

    vTaskStartScheduler();

    while(1);
    return 0;
}
```

The following should be in the CMakeLists.txt

```

cmake_minimum_required(VERSION 3.13)

SET(FREERTOS_KERNEL_PATH ${CMAKE_CURRENT_SOURCE_DIR}/lib/FreeRTOS-Kernel)

include(pico_sdk_import.cmake)
include(${FREERTOS_KERNEL_PATH}/portable/ThirdParty/GCC/RP2040/FreeRTOS_Ker
nel_import.cmake)
include_directories(${CMAKE_SOURCE_DIR})

project(Project C CXX ASM)
```

```
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

# Initialize the Pico SDK
pico_sdk_init()

add_subdirectory(dht)

# Add the C source file
add_executable(Project
    Project.c
)

# Enable UART and USB for stdio
pico_enable_stdio_uart(Project 1)
pico_enable_stdio_usb(Project 1)

# Create map/bin/hex/uf2 files
pico_add_extra_outputs(Project)

# Link with the Pico SDK libraries and FreeRTOS
target_link_libraries(Project
    pico_stdlib           # For standard Pico SDK functionality
    FreeRTOS-Kernel       # For the FreeRTOS kernel
    FreeRTOS-Kernel-Heap4 # For heap memory management in FreeRTOS (or
choose another heap type)
    dht
)
```

pico_sdk_import.cmake

```
# This is a copy of <PICO_SDK_PATH>/external/pico_sdk_import.cmake

# This can be dropped into an external project to help locate this SDK
# It should be include()ed prior to project()

if (DEFINED ENV{PICO_SDK_PATH} AND (NOT PICO_SDK_PATH))
    set(PICO_SDK_PATH $ENV{PICO_SDK_PATH})
    message("Using PICO_SDK_PATH from environment ('${PICO_SDK_PATH}')")
endif ()

if (DEFINED ENV{PICO_SDK_FETCH_FROM_GIT} AND (NOT PICO_SDK_FETCH_FROM_GIT))
    set(PICO_SDK_FETCH_FROM_GIT $ENV{PICO_SDK_FETCH_FROM_GIT})
    message("Using PICO_SDK_FETCH_FROM_GIT from environment
('${PICO_SDK_FETCH_FROM_GIT}')")
endif ()

if (DEFINED ENV{PICO_SDK_FETCH_FROM_GIT_PATH} AND (NOT
PICO_SDK_FETCH_FROM_GIT_PATH))
    set(PICO_SDK_FETCH_FROM_GIT_PATH $ENV{PICO_SDK_FETCH_FROM_GIT_PATH})
    message("Using PICO_SDK_FETCH_FROM_GIT_PATH from environment
('${PICO_SDK_FETCH_FROM_GIT_PATH}')")
endif ()

set(PICO_SDK_PATH "${PICO_SDK_PATH}" CACHE PATH "Path to the Raspberry Pi
Pico SDK")
set(PICO_SDK_FETCH_FROM_GIT "${PICO_SDK_FETCH_FROM_GIT}" CACHE BOOL "Set to
ON to fetch copy of SDK from git if not otherwise locatable")
set(PICO_SDK_FETCH_FROM_GIT_PATH "${PICO_SDK_FETCH_FROM_GIT_PATH}" CACHE
FILEPATH "location to download SDK")

if (NOT PICO_SDK_PATH)
    if (PICO_SDK_FETCH_FROM_GIT)
        include(FetchContent)
        set(FETCHCONTENT_BASE_DIR_SAVE ${FETCHCONTENT_BASE_DIR})
        if (PICO_SDK_FETCH_FROM_GIT_PATH)
            get_filename_component(FETCHCONTENT_BASE_DIR
"${PICO_SDK_FETCH_FROM_GIT_PATH}" REALPATH BASE_DIR "${CMAKE_SOURCE_DIR}")
        endif ()
        # GIT_SUBMODULES_RECURSE was added in 3.17
        if (${CMAKE_VERSION} VERSION_GREATER_EQUAL "3.17.0")
            FetchContent_Declare(
                    pico_sdk
                    GIT_REPOSITORY https://github.com/raspberrypi/pico-sdk
                    GIT_TAG master
                    GIT_SUBMODULES_RECURSE FALSE
            )
        else ()
            FetchContent_Declare(
                    pico_sdk
                    GIT_REPOSITORY https://github.com/raspberrypi/pico-sdk
                    GIT_TAG master
            )
        endif ()

        if (NOT pico_sdk)
            message("Downloading Raspberry Pi Pico SDK")
            FetchContent_Populate(pico_sdk)
            set(PICO_SDK_PATH ${pico_sdk_SOURCE_DIR})
        endif ()
        set(FETCHCONTENT_BASE_DIR ${FETCHCONTENT_BASE_DIR_SAVE})
    else ()
        message(FATAL_ERROR
                "SDK location was not specified. Please set PICO_SDK_PATH
or set PICO_SDK_FETCH_FROM_GIT to on to fetch from git."
                )
    endif ()
endif ()

get_filename_component(PICO_SDK_PATH "${PICO_SDK_PATH}" REALPATH BASE_DIR
"${CMAKE_BINARY_DIR}")
if (NOT EXISTS ${PICO_SDK_PATH})
    message(FATAL_ERROR "Directory '${PICO_SDK_PATH}' not found")
endif ()
```

```
set(PICO_SDK_INIT_CMAKE_FILE ${PICO_SDK_PATH}/pico_sdk_init.cmake)
if (NOT EXISTS ${PICO_SDK_INIT_CMAKE_FILE})
    message(FATAL_ERROR "Directory '${PICO_SDK_PATH}' does not appear to
contain the Raspberry Pi Pico SDK")
endif ()

set(PICO_SDK_PATH ${PICO_SDK_PATH} CACHE PATH "Path to the Raspberry Pi
Pico SDK" FORCE)

include(${PICO_SDK_INIT_CMAKE_FILE})
```