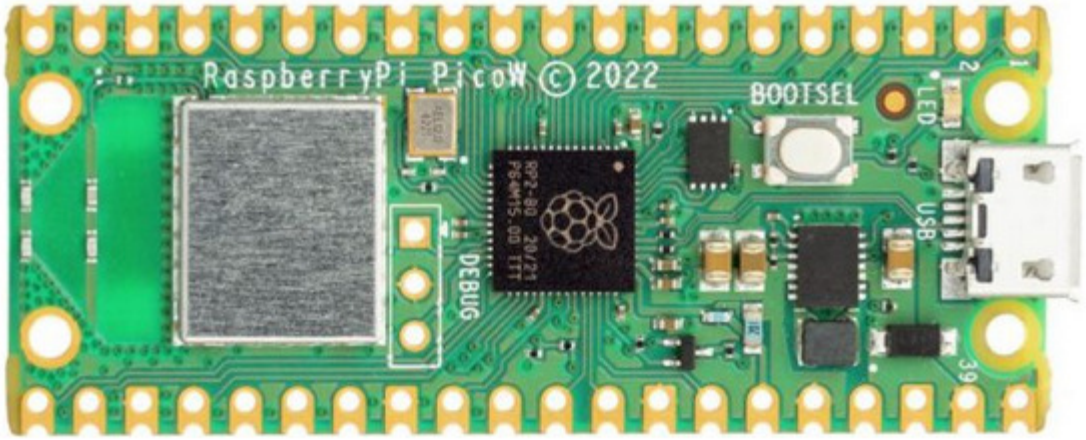
Pico-Flash-Config



Library allowing you to save your project configuration in Pico's flash memory

Firmware Version 1.00
User Guide
Updated April 23rd, 2024

IMPORTANT :

This User Guide is about Pico-Flash-Config Firmware Version 1.00 from Andre St. Louys.
To get the full potential of this library, you must not simply clone the GitHub repository. Read this User Guide to get the most from this library.

Join our Pico-Flash-Config discussion group on:
https://github.com/astlouys/Pico-Flash-Config/discussions

Pico-Flash-Config User Guide

Index

Index 2
Introduction 3
Setup procedure 4
How to use 5

Introduction

The Pico-Flash-Config library is a C language utility that will allow you to save data in Pico's flash memory and retrieve this data whenever needed, even after a power failure and / or power cycle (Off / On). This is not a standalone utility but rather a « library » that must be linked to your program's source code.
NOTE: It is not a « library » in the technical definition, but nonetheless, it should be easy for you to configure it and add it to you current project. (The term « library » will still be used throughout this User Guide).

Take note that the Pico's flash memory lifecycle is estimated to more or less 100,000 write cycles. To write a configuration and / or save some data once in a while, this is more than enough. However, if you plan to make many writes a second, this library may not be a good starting point since there is no wear-leveling algorithm included.

Instead of simply copying the two source files (« Pico-Flash-Config.c » and « Pico-Flash-Config.h ») directly into your project directory, we will create a directory especially for the two files and we will generate a symbolic link to them from your program directory instead. This way, for your project, it will be « like if » the two files would be directly in your project directory, but at the same time, if you use the library among many other projects, once you change / modify / improve / update / debug the files in their own directory, the change becomes instantly available for all other projects that use them.

The instructions below will guide you through the whole setup procedure. Those instructions assume that you setup your development environment following the recommended directory names and hierarchy.

Setup procedure

Follow this procedure to properly setup the library on your system:

1) Create the directory « /home/pi/pico/Pico-Flash-Config »
```
mkdir /home/pi/pico/Pico-Flash-Config
```

2) Copy the files « Pico-Flash-Config.c » and « Pico-Flash-Config.h » to the directory you just created.

3) From you project directory, create two symbolic links to the two source files:
```
ln -s ../Pico-Flash-Config/Pico-Flash-Config.c
ln -s ../Pico-Flash-Config/Pico-Flash-Config.h
```

4) In your project C source file, add the « Pico-Flash-Config.h » include file:
```
#include "Pico-Flash-Config.h"
```

5) In your project C source file, create the structure that will contain the data you want to save to flash memory. I suggest the structure name « flash_config » and a variable name of « FlashConfig » for this variable, but any name can do the job...

6) What is important, however, is that the last member of this structure must be an unsigned 16-bit variable (« UINT16 ») and must be named « Checksum », or « Crc », or the like. In other words, the last UINT16 of the structure is reserved for internal use (reserved). Also, this variable must be at the beginning of a 32-bit boundary in the structure.

7) It is important to remember that writing to flash memory with the Raspberry Pi Pico is limited to a chunk of a maximum of one sector size (which is 4096 bytes or 0x1000). If you want to save more data, you will need to split the data in smaller chunks (maximum of 4096 bytes each).

8) In your project CMakeLists.txt file, in the section « add_executable », add the source file « Pico-Flash-Config.c ».

9) Also in your project CMakeLists.txt file, in the section « target_link_libraries », add the following if they are not already there: « hardware_flash », « hardware_irq » and « pico_multicore ».

How to use

You will usually make a call to « flash_read_config() » when your program starts, to retrieve the configuration as it has been saved to flash memory. You will have to cast your structure pointer to a unsigned int 8 pointer « UINT8 * » in your call:

```
flash_read_config(FLASH_CONFIG_OFFSET, (UINT8*)&FlashConfig, sizeof(FlashConfig))
```

When coming back from the « flash_read_config() » call, you may want to confirm the integrity of the configuration read back. To do so, you will want to make sure that the Checksum read back from flash corresponds to the actual checksum computed on the data. The function « flash_extract_crc() » will return the CRC saved to flash (you can also, more easily, used the UINT16 variable that you declared as the last member of your configuration structure. You will compare this checksum with the one calculated with the configuration data with the help of the « util_crc16() » function. If both checksum match, it means that the configuration read back from flash is OK. Be aware that when using the « util_crc16() » function, the DataSize must be 2 bytes (one « UINT16 ») less than your structure size, in order to exclude the « CRC » structure member from the crc calculation.

How to use the « flash_save_config() » depends of the algorithms used in your program. You may want to save a new copy of the configuration as soon as something changes in its content. You may also setup a callback function that is called, say, every hour, and that will compute the current checksum of your configuration structure. If something has changed, the computed checksum will not be the same as the one that is the last member of the structure (and that has been read back from flash on the last « flash_read_config() »). If so (if both checksums do not match), it means that something has changed in the current configuration and you may want to save the new configuration. I want to repeat once more that you should not use those functions to write to flash many times a second since no « wear levelling » algorithm has been implemented and you will rapidly reach the expected write cycle life of your Pico (more or less 100,000 writes cycles).

The « flash_display() » function is relatively self explanatory (you may also take a look at the comments in the source code) and the same goes for the function « util_display_data() ».

« flash_erase() » and « flash_write() » should usually not be used outside of the « flash_config.c » file.

If you want to see how to use those functions, you may want to take a look at the « Pico-RGB-Matrix » and / or « Pico-Green-Clock » in my repositories. I do not use the exact same library, but you should easily refer to the same function names to see how everything goes...