Functions

- void flash_range_erase (uint32_t flash_offs, size_t count)
- Erase areas of flash.
- void flash_range_program (uint32_t flash_offs, const uint8_t *data, size_t count)
- Program flash.
- void flash_get_unique_id (uint8_t *id_out)
- Get flash unique 64 bit identifier.
- void flash_do_cmd (const uint8_t *txbuf, uint8_t *rxbuf, size_t count)
- Execute bidirectional flash command.

Detailed Description

Low level flash programming and erase API

Note these functions are *unsafe* if you are using both cores, and the other is executing from flash concurrently with the operation. In this could be the case, you must perform your own synchronisation to make sure that no XIP accesses take place during flash programming. One option is to use the lockout functions.

Likewise they are *unsafe* if you have interrupt handlers or an interrupt vector table in flash, so you must disable interrupts before calling in this case.

If PICO_NO_FLASH=1 is not defined (i.e. if the program is built to run from flash) then these functions will make a static copy of the second stage bootloader in SRAM, and use this to reenter execute-in-place mode after programming or erasing flash, so that they can safely be called from flash-resident code.

Example

```
#include <stdio.h>
#include <stdlib.h>

#include "pico/stdlib.h"
#include "hardware/flash.h"

// We're going to erase and reprogram a region 256k from the start of flash.
// Once done, we can access this at XIP_BASE + 256k.
#define FLASH_TARGET_OFFSET (256 * 1024)

const uint8_t *flash_target_contents = (const uint8_t *) (XIP_BASE +
FLASH_TARGET_OFFSET);

void print_buf(const uint8_t *buf, size_t len) {
for (size_t i = 0; i < len; ++i) {
printf("%02x", buf[i]);
if (i % 16 == 15)
printf("\n");
else
printf(" ");
}
}

int main() {
stdio_init_all();
uint8_t random_data[FLASH_PAGE_SIZE];
for (int i = 0; i < FLASH_PAGE_SIZE; ++i)
random_data[i] = rand() >> 16;
```

```
printf("Generated random data:\n");
print_buf(random_data, FLASH_PAGE_SIZE);

// Note that a whole number of sectors must be erased at a time.
printf("\nErasing target region...\n");
flash_range_erase(FLASH_TARGET_OFFSET, FLASH_SECTOR_SIZE);
printf("Done. Read back target region:\n");
print_buf(flash_target_contents, FLASH_PAGE_SIZE);

printf("\nProgramming target region...\n");
flash_range_program(FLASH_TARGET_OFFSET, random_data, FLASH_PAGE_SIZE);
printf("Done. Read back target region:\n");
print_buf(flash_target_contents, FLASH_PAGE_SIZE);

bool mismatch = false;
for (int i = 0; i < FLASH_PAGE_SIZE; ++i) {
if (random_data[i] != flash_target_contents[i])
mismatch = true;
}
if (mismatch)
printf("Programming failed!\n");
else
printf("Programming successful!\n");
}
```

Function Documentation

◆ flash_do_cmd()

void flash_do_cmd	(	const uint8_t *	*txbuf*,
		uint8_t *	*rxbuf*,
		size_t	*count*
	)		

Execute bidirectional flash command.

Low-level function to execute a serial command on a flash device attached to the QSPI interface. Bytes are simultaneously transmitted and received from txbuf and to rxbuf. Therefore, both buffers must be the same length, count, which is the length of the overall transaction. This is useful for reading metadata from the flash chip, such as device ID or SFDP parameters.

The XIP cache is flushed following each command, in case flash state has been modified. Like other hardware_flash functions, the flash is not accessible for execute-in-place transfers whilst the command is in progress, so entering a flash-resident interrupt handler or executing flash code on the second core concurrently will be fatal. To avoid these pitfalls it is recommended that this function only be used to extract flash metadata during startup, before the main application begins to run: see the implementation of pico_get_unique_id() for an example of this.

Parameters

txbuf	Pointer to a byte buffer which will be transmitted to the flash
rxbuf	Pointer to a byte buffer where data received from the flash will be written. txbuf and rxbuf may be the same buffer.

count Length in bytes of txbuf and of rxbuf

◆ flash_get_unique_id()

void flash_get_unique_id (uint8_t * *id_out*)

Get flash unique 64 bit identifier.

Use a standard 4Bh RUID instruction to retrieve the 64 bit unique identifier from a flash device attached to the QSPI interface. Since there is a 1:1 association between the MCU and this flash, this also serves as a unique identifier for the board.

Parameters

id_out Pointer to an 8-byte buffer to which the ID will be written

◆ flash_range_erase()

void flash_range_erase (uint32_t *flash_offs*,
size_t *count*
)

Erase areas of flash.

Parameters

flash_offs Offset into flash, in bytes, to start the erase. Must be aligned to a 4096-byte flash sector.
count Number of bytes to be erased. Must be a multiple of 4096 bytes (one sector).

◆ flash_range_program()

void flash_range_program (uint32_t *flash_offs*,
const uint8_t * *data*,
size_t *count*
)

Program flash.

Parameters

flash_offs Flash address of the first byte to be programmed. Must be aligned to a 256-byte flash page.
data Pointer to the data to program into flash
count Number of bytes to program. Must be a multiple of 256 bytes (one page).