

Embedded pool

Day 05 : EEPROM

contact@42chips.fr

Summary: Because writing too many times at the same spot is what made cars crash

Chapter I

Introduction

An EEPROM (Electrically Erasable Programmable Read-Only Memory) is a type of non-volatile memory that can be used to store data on a microcontroller.

It can be written to and erased multiple times, and retains its data when the power is turned off.

EEPROMs are useful for storing data that needs to be retained even when the microcontroller is not powered, such as configuration settings or calibration data.

They are slower and have less capacity than other types of memory, such as SRAM or flash memory, but are still useful in a variety of applications.

Chapter II


General instructions

Unless explicitly stated otherwise, the following instructions will be valid for all assignments.

- The language used for this project is C.
- It is not necessary to code according to the 42 norm.
- The exercises are ordered very precisely from the simplest to the most complex. Under no circumstances will we consider or evaluate a complex exercise if a simpler one is not perfectly successful.
- You must not leave any files other than those explicitly specified by the exercise instructions in your directory during peer evaluation.
- All technical answers to your questions can be found in the **datasheets** or on the Internet. It is up to you to use and abuse these resources to understand how to complete your exercise.
- You must use the datasheet of the microcontroller provided to you and comment on the important parts of your program by indicating where you found the clues in the document, and if necessary, explaining your approach. Don't write long blocks of text, keep it clear.
- Do you have a question? Ask your neighbor to the right or left. You can ask in the dedicated channel on the Piscine's Discord, or as a last resort, ask a staff member.

Chapter III


Tutorial

	Exercise 00
Memorizing state	
Turn-in directory : <code>ex00/</code>	
Files to turn in : <code>Makefile</code> , <code>*.c</code> , <code>*.h</code>	
Allowed functions : <code>avr/io.h</code> , <code>avr/eeprom.h</code>	
Notes : n/a	

- use the EEPROM to save and restore the state of a counter.
- use the SW1 button to increase the count.
- use the LEDs D1, D2, D3, D4 on the board to show the current state of the counter.



Do not assume the EEPROM is set at 0 when you didn't write to it yet.

	Exercise 01
Multiplex	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Makefile , *.c , *.h	
Allowed functions : avr/io.h , avr/eeprom.h	
Notes : n/a	


- use the EEPROM to save and restore the state of 4 counters.
- use the EEPROM to save and restore the current selected counter.
- use the button SW1 to increase the count.
- use the button SW2 to select a counter.
- use the LEDs D1, D2, D3, D4 to show the current state of the selected counter.



It might be a good idea to look what a magic number is ;)

Chapter IV

Griding before the final boss

	Exercise 02
The only thing I know for real	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <i>Makefile</i> , <i>*.c</i> , <i>*.h</i>	
Allowed functions : <i>avr/io.h</i> , <i>avr/eeprom.h</i>	
Notes : n/a	


For this exercise you must write a magic number in front of the block of memory that you write in the EEPROM. It is needed to know that this data has already been written by you. Do not rewrite the bytes of the data block if the value is the same.

With this in mind write the functions:

```
bool safe_eeprom_read(void *buffer, size_t offset, size_t length);
bool safe_eeprom_write(void *buffer, size_t offset, size_t length);
```



The number of write cycles on an EEPROM is limited, but not the number of reads.

	Exercise 03
Bobby; DROP TABLE	
Turn-in directory : <i>ex03/</i>	
Files to turn in : Makefile, *.c, *.h	
Allowed functions : avr/io.h, avr/eeprom.h	
Notes : n/a	

You must now write 3 functions which will allow you, like a dictionary, to write, reserve and [free memory spaces](#) in the EEPROM.

You must clean up the deleted memory spaces to be able to reallocate them.

```
bool eepromalloc_write(uint16_t id, void *buffer, uint16_t length)
bool eepromalloc_read(uint16_t id, void *buffer, uint16_t length)
bool eepromalloc_free(uint16_t id)
```




You can assume you are the only one to use the EEPROM. If you don't have any space left or it doesn't fit in the EEPROM when allocating simply return false.



No need to say that you can never forget or corrupt data that hasn't been freed.

Chapter V

FINAL BOSS

	Exercise 04
EEPROMalloc	
Turn-in directory : <i>ex04/</i>	
Files to turn in : Makefile , *.c , *.h	
Allowed functions : avr/io.h , avr/eeprom.h	
Notes : n/a	

Write a command line interface on the UART port of your microcontroller. It stores a pair of key values strings which cannot be lost on reboots (includes powering off the devkit). The keys and values are delimited by double quotes ("). It takes 3 commands :

- **READ**: Given 1 key, recovers the value associated. if not found returns a newline.
- **WRITE**: Given 1 key, 1 value, try to store them and reports the status:
 - **true**: Done.
 - **false**: No space left on device.
- **FORGET**: Given 1 key, removes key value pair if found and reports the status:
 - **true**: Done.
 - **false**: Not found.



The exercise doesn't ask for an implementation of malloc. Simply a way to write on bits of memory without erasing what's already stored there.
Do not make it harder for yourself.


```
> READ "key-1"

> WRITE "key-1" "dunno"
Done.
> READ "key-1"
"dunno"
> FORGET "key-2"
Not found.
> FORGET "key-1"
Done.
> READ "key-1"
```

```
> READ "key-1"

> WRITE "key-1" "very long string, maybe something from wikipedia ?"
Done.
> WRITE "key-2" "very long string, maybe something from wikipedia ?"
Done.
...
> WRITE "key-XXX" "very long string, maybe something from wikipedia ?"
No space left.
```