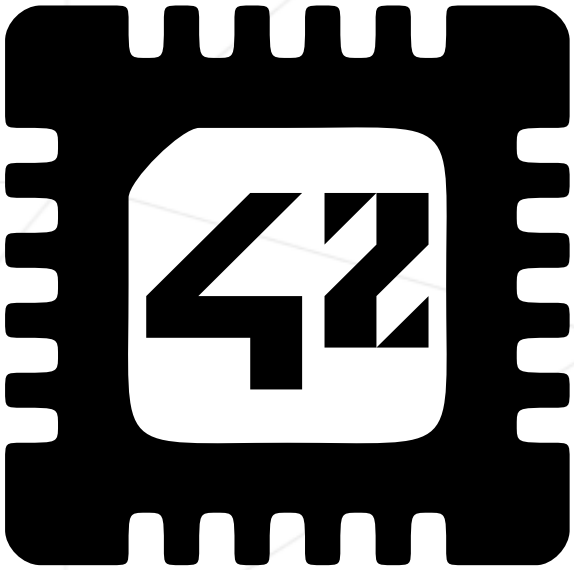


Embedded pool

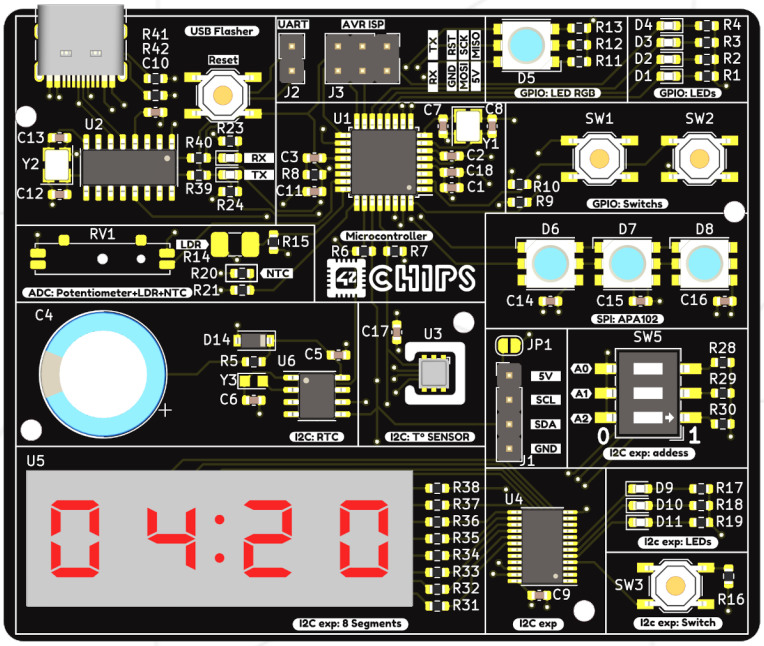
Day00: First Programs

contact@42chips.fr

Summary: Once upon a time, there was a little spectrum...

Chapter I

Prologue



For the entire piscine, you will use the provided devkit. It's based the ATMega328p microchip, a really nice microprocessor, completely in tune with the times

Exercises will be done in C. You will use the `avr-gcc` cross-compiler for the AVR architecture.

To write to the devkit's flash memory, you will use `avr-objcopy` & `avrdude`. You will use them with the `arduino` programmer with a `115200` baud rate.

The 2 most important documents for the whole piscine are:

- the ATmega328p's datasheet (653 pages of pure bliss);
- the devkit's schema.

Good luck to everyone, despite the intensity of the piscine, try not to get too wired! We're sure you'll realize your full potential!

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

Setup

	Exercise 00
Makefile	
Turn-in directory : $ex00/$	
Files to turn in : `Makefile, *.c, *.h`	
Allowed functions : `avr-gcc, avr-objcopy, avrdude`	
Notes : `n/a`	

- The `main.c` file should contain a main() program that does not contain any instructions.
- Write a `Makefile` file with the `all` rule that executes the `hex` and then `flash` rules.
- The `hex` rule compiles the `main.c` file into `main.bin` with an F_CPU variable to select the microcontroller frequency. Then it generates the `main.hex` file from the `main.bin` file.
- The `flash` rule copies the `main.hex` file into the microcontroller's flash memory.
- The `Makefile` must also implement the `clean` rule that deletes the `main.hex` and `main.bin` files.

Chapter IV

Let there be light

	Exercise 01
A glimmer of hope	
Turn-in directory : *ex*01/	
Files to turn in : `Makefile, *.c, *.h`	
Allowed functions : `avr/io.h`	
Notes : `n/a`	

- You must write a program that turns on the LED D1 (PB0).
- You must use only AVR registers (DDRX, PORTX, PINX).

`You must explain the function and values assigned to the registers in comments every time!`

	Exercise 02
	Lumos
Turn-in directory : $ex02/$	
Files to turn in : `Makefile, *.c, *.h`	
Allowed functions : `avr/io.h, util/delay.h`	
Notes : `n/a`	

- You must write a program that turns on LED D1 (PB0) when button SW1 (PD2) is pressed.
- When the button is released, the LED turns off.
- You must use only AVR registers (DDRX, PORTX, PINX).

	Exercise 03
	Day, Night, Day, Night
Turn-in directory :	*ex*03/
Files to turn in :	`Makefile, *.c, *.h`
Allowed functions :	`avr/io.h, util/delay.h`
Notes :	`n/a`

- You must write a program that reverses the state of the PB0 LED each time the SW1 button (PD2) changes from the released state to the pressed state.
- You must use only AVR registers (DDRX, PORTX, PINX).

`Be careful of bounce effects!`

Chapter V

Bonus: It's not rocket science !

	Exercise 04
Binary Counter	
Turn-in directory : *ex*04/	
Files to turn in : `Makefile, *.c, *.h`	
Allowed functions : `avr/io.h, util/delay.h`	
Notes : `n/a`	

- You must write a program that:
  - increments a value each time you press button SW1
  - decrements a value each time you press button SW2
  - displays this value in binary on LEDs D1 D2 D3 and D4 permanently.
- You must use only the AVR registers (DDRX, PORTX, PINX)

`If the 4 LEDs were on GPIO PB0, PB1, PB2, PB3, this exercise would be simpler.`
`Unfortunately, LED D4 is on PB4 instead of PB3 because PB3 is used for something else.`
`You will have to manipulate bits.`