



Embedded pool

Module01 : Timers

`contact@42chips.fr`

Summary: Tic Tac, Tic Tac, Tic Tac.

Version: 1

Chapter I

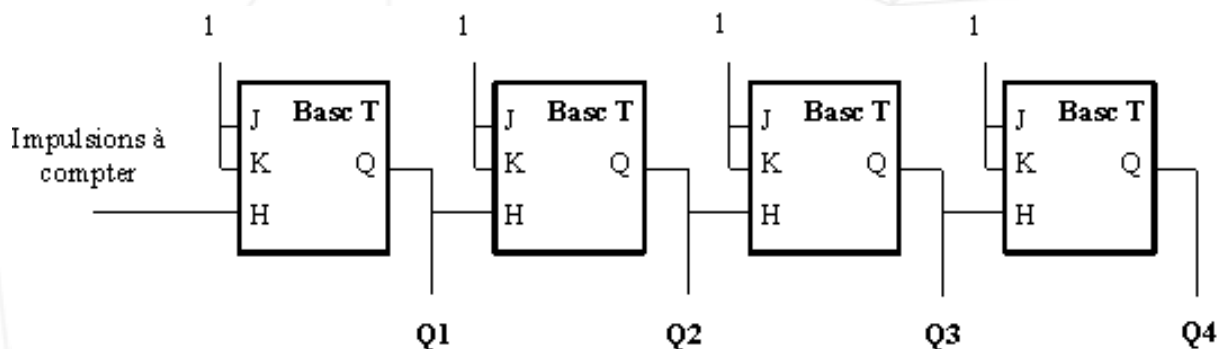
Introduction

In electronics, a counter is a digital integrated circuit designed to count the number of pulses applied to its input.

It is composed of a certain number of [D](#), [T](#) or [JK flip-flops](#).

The simplest counter is obtained by cascading a series of T flip-flops, with the signal to be counted being applied to the input of the first flip-flop; the output of this flip-flop drives the input of the second flip-flop and so on.

The result of the counting appears in the form of a binary number, with the first flip-flop indicating the least significant digit.



The capacity of the counter is the maximum number of pulses it can totalize. It is equal to 2^N for a binary counter.

If the capacity is exceeded, the counter returns to 0 and starts counting again.

But enough joking around, let's now see what interests us the most.
How to do it in [minecraft](#).

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

Tic & tac

	Exercise 00
	Blinker
	Turn-in directory : <i>ex00/</i>
	Files to turn in : Makefile , *.c , *.h
	Allowed functions : avr/io.h


- You must write a program that turns on and off the LED D2 (PB1) at a frequency of around 1Hz.
- You must write code that allows you to wait for several hundred milliseconds and which will be inserted into the infinite loop of the program (no hardware TIMER).
- The change in state of the LED must be made with a single [bitwise operation](#), do not use a condition (if else).
- You must use only the AVR registers (DDRX, PORTX, PINX).



1Hz == (on for 0.5 sec and off for 0.5 sec).
The goal is to light an LED up. No need to have an atomic clock's precision here.




You must explain every value assigned to the registers. The exercise will be considered invalid if your main returns.

	Exercise 01
Timer1	
Turn-in directory : <i>ex01/</i>	
Files to turn in : Makefile, *.c, *.h	
Allowed functions : avr/io.h	


- You must write a program that turns on and off the LED D2 (PB1) at a frequency of 1Hz.
- You must configure the registers of the **Timer1** to control the LED.
- The infinite loop of the program must remain empty.
- And you must not use PORTX.



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

	Exercise 02
Duty cycle	
Turn-in directory : <i>ex02/</i>	
Files to turn in : Makefile , *.c , *.h	
Allowed functions : avr/io.h	

- You are required to write a program that turns on and off LED D2 (PB1) at a frequency of 1Hz with a **duty cycle** of 10%.
- You must configure the registers of **Timer1** to control the LED.
- The infinite loop of your program must remain empty.
- You must not use PORTX.

	Exercise 03
The cycle of life	
Turn-in directory : <i>ex03/</i>	
Files to turn in : Makefile , *.c , *.h	
Allowed functions : avr/io.h , util/delay.h	

- You are required to write a program that turns on and off LED D2 (PB1) at a frequency of 1Hz with a variable **duty cycle** going from 10 to 100%.
- You must configure the registers of **Timer1** to control the LED.
- The duty cycle must be managed as follow:
 - Pressing button SW1 increments the duty cycle by 10 %.
 - Pressing button SW2 decrements the duty cycle by 10 %.



delay is only allowed for debouncing the buttons
Any other usage will invalidate this exercise