# Snake Game on Oscilloscope

Hugo Cardoso
up201907840@up.pt

Tiago Sousa
up201907205@up.pt

***Abstract*—This report details the process of elaborating a system capable of playing the classic Snake game developed resorting to an oscilloscope to display graphics. The project was developed using an Arduino Uno and a digital oscilloscope with the display mode set to XY. A fixed priority preemptive micro-kernel was implemented to ensure stable and periodic refresh rate. A schedulability analysis of the various tasks was conducted, in order to guarantee the system met the real-time requirements.**

## I. Introduction

The project presented in this report consists of a classic Snake game that is displayed on an oscilloscope using an Arduino Uno microcontroller and an MCP4822 Digital-to-Analog Converter (DAC). The main objective of this project is to create a real-time interactive system with fluid presentation and good gameplay. The game starts with a snake that moves across the game board and the player must direction the snake so that it eats randomly placed fruits to grow, where each fruit eaten increases its length. User inputs are captured through four push buttons that control the direction of the snake's movement. The game ends when the snake collides with itself or has reached the maximum size.

## II. Hardware

For the development of the project the following material was used:

1) 1 Tektronix TDS 1002B Oscilloscope
2) 1 Arduino Uno Rev3
3) 1 MCP4822 DAC
4) 4 Push buttons
5) 4 Diodes

Figure 1 displays the circuit diagram.

## III. Requirements

In order to achieve a fluid gameplay experience the system must be able to produce enough frames so that the player does not detect any artifacts or discontinuations. A frame consists of drawing the board, the snake, and the fruit in the oscilloscope. However, the fruit is the least important part of the frame as it is not crucial for the gameplay experience, as the fruit can be drawn in the oscilloscope with delay or at a lower frequency without breaking the gameplay. Furthermore, the system must also be able to handle the game logic at this frequency, meaning the calculation of the coordinates of the next snake, deducing whether or not the snake collided with itself or has eaten the fruit or if the snake has hit a border and needs to appear on the opposite border with which it collided while also handling the player's input. Typically, a reasonable gameplay experience can be achieved by drawing at least 30 full frames per second. However, during testing we have found that this produces some discontinuous shapes, as a result we have set the minimum frames per second at 40, which implies a frequency of 40Hz and that every task must be triggered and completed at least once every 25ms, which requires a macro-cycle equal or smaller than 25ms. On the one hand, the higher the number of frames per second the smoother the player's experience will be. On the other hand drawing frames requires computational power and the more frames are drawn the less computational power is available to handle the logic of the game. This trade-off must be balanced so that a smooth and consistence experience can be presented.

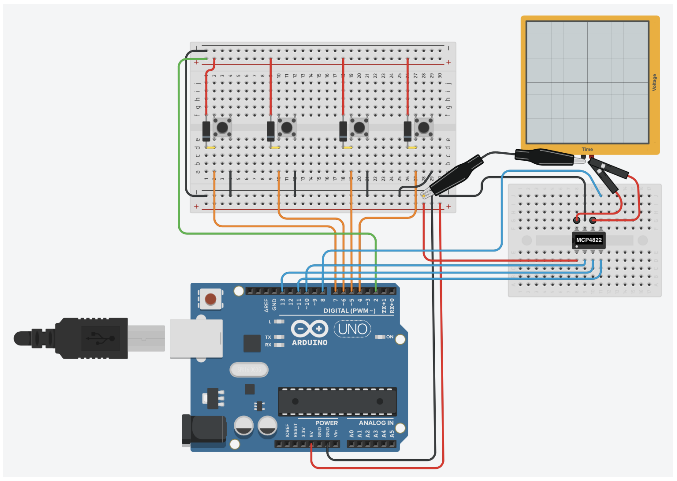

Fig. 1. Circuit Hardware

## IV. Schedulability Analysis and Software Architeture

### A. Tasks

The system operates using a set of five tasks, which are presented in order of their priorities:

1) **inputInterrupt()**: handles the user input;
2) **nextSnake()**: handles the game logic including: updating the coordinates for next position of the snake, based on current direction. Checking if the fruit has been eaten and generating new random coordinates for the fruit if it has been eaten. Checking if the snake is within the board's boundaries transporting to the opposite border if

it goes out of boundary. Resetting the game if the snake has eaten itself;
3) **drawSnake()**: draws the snake on the oscilloscope;
4) **drawSquare()**: draws the game board boundaries on the oscilloscope;
5) **drawFruit()**: draws the fruit at its current location on the oscilloscope.

The inputInterrupt is an interrupt that occurs once the player has pressed any of the four buttons that change the direction of the snake according to the button pressed. This task can be seen as the highest priority task for the sake of schedulability analysis as it behaves as an aperiodic task that has a higher priority than all others, considering that the kernel is preemptive and priority based, although this task is not scheduled, the same way the other four are. The nextSnake task has the second highest priority, as without it the snake remains in the same coordinates, rendering the game unresponsive. The drawSnake and drawSquare are equally important as the game cannot function properly without one or the other. As previously explained in III, the drawFruit task is the least important as not drawing it after everything else has been drawn does not disrupt gameplay (i.e occasionally missing deadlines for drawFruit) or make the the player lose, whereas not drawing the snake, the boundaries, missing inputs or not calculating the next coordinates of the snake may make the player lose or make the game unresponsive.

## B. *Task Scheduling*

A preemptive kernel was implemented, provided in the course's documents, where tasks are scheduled to activate in well-defined instants. The kernel works by attaching an interrupt to a hardware timer which is compared to a register so that this interrupt triggers at a frequency of 2kHz. During this interrupt the kernel schedules the next tasks according to their period, meaning if the period of the task has already elapsed then it sets a flag for execution, otherwise it decreases the delay (which is previously set to the value of the period) of the task by one. After scheduling the tasks, the tasks are executed based on their priority, where the first task in the array of tasks has the highest priority. Note that the kernel also allows preemption, meaning if a lower priority task is executing and a higher priority task is flagged for execution, the highest priority task will be executed, preempting the lower priority task. The scheduler ensures that tasks are executed on time to meet the game's real-time requirements.

## C. *Drawing on the oscilloscope*

An analog oscilloscope can be used to draw shapes if switched to the "XY" mode, where the voltage applied channel 1 is the coordinate X and the voltage applied to channel 2 is the Y coordinate of a given point. To write synchronously to each channel, a DAC paired with the serial peripheral interface (SPI) communication protocol is used. Furthermore, both voltages were applied to each channel at the same time to ensure that the coordinate written is precise and did not create artifacts due to previously buffered coordinates. This

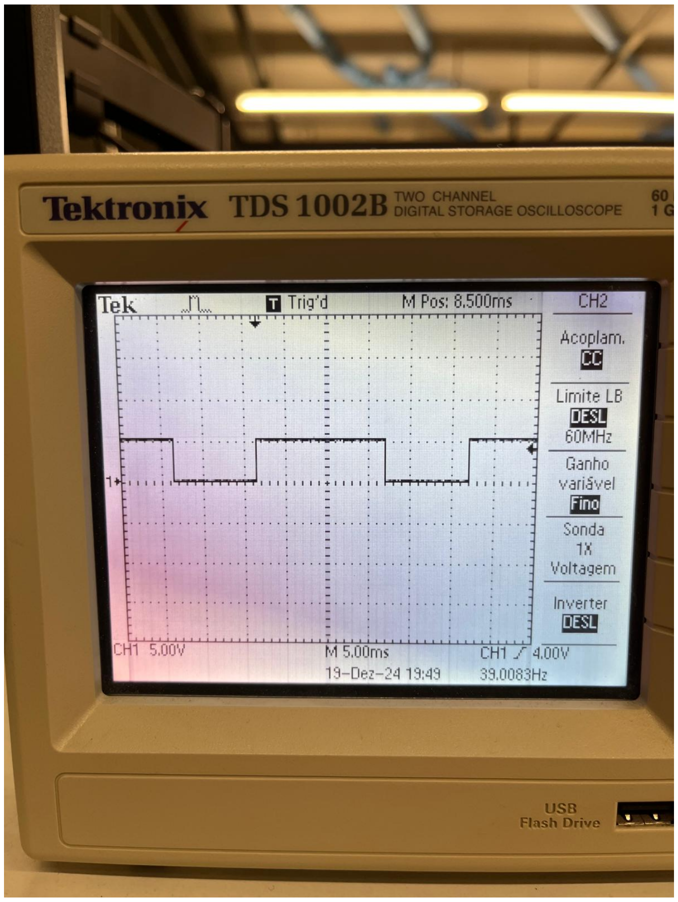

Fig. 2. Example of the time measuring process for the drawSquare task

is accomplished through the use of the inherent buffer of the MCP4822 DAC chip and by controlling the latch DAC (LDAC) pin[1], which when given a pulse, applies the data of the buffer to the outputs of the chip. This is a shared resource among tasks as multiple tasks require access to this output.

## D. *Schedulability*

The time each task takes to execute was measured in isolation and in their worst case scenario (i.e. the path with longest execution time for that task). The measuring methodology was the same for them all, which was setting an output pin to a high value before the task executes and setting that same to a low value when the task has ended and then using an oscilloscope connected to that pin to determine the time in which that pin is set to high, which is the time the task took to execute, an example of this process can be seen in Figure 2. However, a different process was used for the inputInterrupt task, which was measured resorting to stamping the time it started execution and the time it ended execution and subtracting the two timestamps. In Table I the execution times gathered are displayed.

Note that the drawSnake task and the nextSnake task depend on the length of the snake, which starts at 20 and each fruit

[1]MCP4822's datasheet

TABLE I
MEASURED TIMES AND PERIOD FOR EACH TASK

| Task | Period (T) | Execution Time (C) |
|---|---|---|
| inputInterrupt | aperiodic | 0.24ms |
| nextSnake | 25ms | 0.55ms |
| drawSnake | 25ms | 7ms |
| drawSquare | 25ms | 16ms |
| drawFruit | 25ms | 0.05ms |

eaten increases it's length by 5. Since the maximum score is 29 fruits the maximum length is 165, considering the next fruit eaten would increase the score to 170 but would also reset the game due to the maximum score reached. If the score was set higher, the execution time of the drawSnake and nextSnake tasks would increase, making the tasks be unschedulable. This gameplay feature is a trade-off that must be balanced, as a higher score would increase the play time and the gameplay quality but would also leave less time for game logic computations. Furthermore, the maximum score could be increased if lower frame rates are accepted because the period of the tasks would increase creating more time for game logic computations. Both schedulability analysis and the results presented in Table I are made considering the worst case scenario, meaning that during execution time measuring, the length of the snake will be the maximum length of 165. Furthermore, the execution time increases exponentially with the length of the snake (the execution time of the drawSnake and nextSnake tasks is 0.9/1.75/3.4ms and 0.105/0.165/0.330ms respectively when the length is set to 20/40/100), which means that the worst case scenario (which only happens once the player is one fruit away from maximum score) is significantly more demanding in computational time than the average runtime. Despite the fact that the inputInterrupt can, in theory, be triggered as many times as the user can press them in a 25ms period (excluding physical limitations), we assumed the user would need observe the direction before pressing another button or would simply not be quick enough to break this threshold[2], which implies that this tasked is, at most, triggered once per period of 25ms. As a result the schedulability analysis is done as if this task is periodic with a period of 25ms. Regarding resource sharing: three tasks share the resource that writes to the DAC. Clarifying, this resource writes one point at a time, which means the resource can not be block for the entire duration of a task, it can only be blocked during the writing of a single point, which was measured to be 0.05ms[3]. This resource was implemented resorting to disabling and enabling interrupts, which means that, at most, the tasks drawSnake and drawSquare may be blocked for 0.05ms due to sharing resources with drawFruit. The schedulability was analyzed resorting to the worst case response time for each task and was done as follows:

$$\tau_1 : R_{wc_1}(0) = C_1 = 0.024ms \tag{1}$$

$$\tau_2 : R_{wc_2}(0) = C_1 + C_2 = 0.024 + 0.55 = 0.574ms \tag{2}$$

$$\begin{aligned} R_{wc_2}(1) = &\left\lceil \frac{0.574}{25} \right\rceil * 0.024 + 0.55 = \\ =&1 * 0.024 + 0.55 = 0.574ms \end{aligned} \tag{3}$$

$$\begin{aligned} &\tau_3 : R_{wc_3}(0) = C_1 + C_2 + C_3 + B_5 = \\ &= 0.024 + 0.55 + 7 + 0.05 = 7.624ms \end{aligned} \tag{4}$$

$$\begin{aligned} R_{wc_3}(1) = &\left\lceil \frac{7.574}{25} \right\rceil * 0.024 + \left\lceil \frac{7.574}{25} \right\rceil * 0.55 \\ &+ 7 + 0.05 = \\ &= 1 * 0.024 + 1 * 0.55 + \\ &+ 7 + 0.05 = 7.624ms \end{aligned} \tag{5}$$

$$\begin{aligned} &\tau_4 : R_{wc_4}(0) = C_1 + C_2 + C_4 + B_5 = \\ &0.024 + 0.55 + 16 + 0.05 = 16.624ms \end{aligned} \tag{6}$$

$$\begin{aligned} R_{wc_4}(1) = &\left\lceil \frac{16.574}{25} \right\rceil * 0.024 + \left\lceil \frac{16.574}{25} \right\rceil * 0.55 + \\ &+ 16 + 0.05 \\ &= 1 * 0.024 + 1 * 0.55 + 16 + 0.05 \\ &= 16.624ms \end{aligned} \tag{7}$$

$$\begin{aligned} \tau_5 : R_{wc_5}(0) =& C_1 + C_2 + C_3 + C_4 + C_5 + B_5 \\ =& 0.024 + 0.55 + 7 + 16 + 0.05 \\ =& 23.624ms \end{aligned} \tag{8}$$

$$\begin{aligned} R_{wc_5}(1) = &\left\lceil \frac{23.624}{25} \right\rceil * 0.024 + \left\lceil \frac{23.624}{25} \right\rceil * 0.55 + \\ &\left\lceil \frac{23.624}{25} \right\rceil * 7 + \left\lceil \frac{23.624}{25} \right\rceil * 16 + 0.05 = \\ =&1 * 0.024 + 1 * 0.55 + \\ &+ 1 * 7 + 1 * 16 + 0.05 \\ =&23.624ms \end{aligned} \tag{9}$$

where Equations 1 validates inputInterrupt's schedulability, Equations 2 and 3 validate nextSnake's schedulability, Equations 4 and 5 validate drawSnake's schedulability, Equations 6 and 7 validate drawSquare's schedulability and Equations 8 and 9 validate drawFruit's schedulability.

[2]According to clickspeedtester, the world record is 9.2 clicks per second (on a traditional mouse) - one click per 109ms -, during testing we managed to get 8 clicks in a second - one click per 125ms -, even if we multiply this by four buttons, the world record would increase to 36.8 clicks per second - one click every 27ms.

[3]roughly the same amount of time as drawFruit, which makes sense, since the drawFruit only draws one point during it's execution

## V. Conclusion

The final result can be observed in a video uploaded to YouTube, where the game is smoothly running and features such as the game ending, the snake eating fruit and growing, the border detection and the user input flawlessly working. Moreover, the schedulability validates the 40Hz refresh rate of the game, under the given maximum score of 29 fruits, which was a goal set to ensure fluid gameplay.

## VI. Group Organization

Both students actively participated during all stages of the project development. Tiago Sousa contributed with 60% and Hugo Cardoso with 40%.