# CPEN412 Winter 2022 Term 2
# University of British Columbia
# Final Project

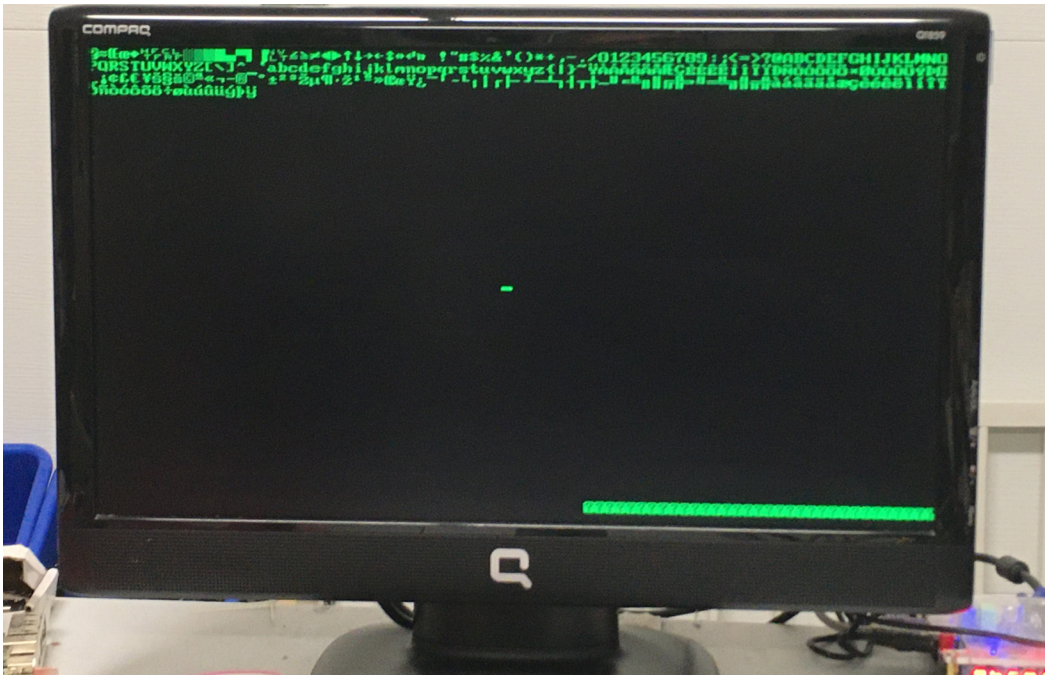
In this final project we will build the retro Snake arcade game. This will include integrating a VGA display core into the project, mapping it to the processor's memory space, writing appropriate functions to operate it, and integrating those software functions into the game's C code, as well as modifying the code as necessary.

## Part 1: Taking a look at the solution and understanding what you need to do

1. Download the "final_project_2022_sem2_solution.sof" solution for this final project from the course website.

2. Now turn on the card and load the solution SOF file to the card using the Quartus standalone programmer.

3. You will see a demo screen, as follows:



4. To play the game press "g" in the hyperterminal.

5. The purpose of the snake game is for the snake (composed of the letters "S") to eat food ("@").Once food is eaten, another food appears at a random place.

6. If while moving the snake runs into itself or into the borders (signified by the characters "#") then it's game over, and a game over screen with some basic animation is shown.

7. The control of the snake is achieved via the hyperterminal. Pressing "w" will make the snake go up, "a" will make it go left, "s" will make it go down, "d" will make it go right. Pressing "p" will pause the game, and pressing it again will unpause. Pressing "q" will quit the game. Once a game has ended, you can press "g" again to start a new game.

**You have to design the contents of the FPGA hardware and software to replicate the behavior of the solution. To complete the lab, follow the steps below.**

## Part 2: Figuring out how to integrate the VGA core

8. Download the student template file:

   final_project_2022_sem2_student_template.zip

9. For this project you will be integrating the a VGA core that is only able to display text.

10. Unzip the student template file. Under the directory "vga80x40" you will find the VGA core. The top level file is "vga80x40.vhd".

11. The documentation of the core is contained in the file "VGA80x40_documentation.pdf". The documentation is short (4 pages) but relatively complete. Remember that if

something is unclear, the core is open source, so you can just look in the source code.

12. This core was originally implemented for Xilinx devices. You will need to port it to work with the DE1-SoC which uses the Altera Cyclone V.

13. This core uses a memory that contains the font (initial value Xilinx file: "lat0-12.coe") and a memory that contains the screen contents (initial value Xilinx file: "ram.coe", which contains the contents of the demo screen you saw in instruction bullet point 3).

14. For your convenience, I have already translated these COE files to MIF files suitable for Altera devices, lat0-12.mif and ram.mif, respectively.

15. Look at the DE1-SoC user manual and schematic. You can also look at the DE1-SoC reference designs included in the DE1-SoC CD. Look in particular for connections to the VGA signals. You will need to figure out how to connect the VGA core to the VGA signals on the DE1-SoC.

16. We will use 1 bit for each of the Red, Green, Blue (RGB) components of the VGA. Since the DE1-SoC has 8 bits for each components, what you need to do is replicate the value of the 1-bit to the 8 bits. For example, if the VGA core emits a 1-bit red component that is "1", what will go out to the 8-bit red bus on the DE1-SoC will be 8'b11111111.

17. The file "vga80x40_test.vhd" would display the demo screen on a Xilinx device.

## Part 3: Completing the project

18. The final project is individual. You are **not allowed** to post public Piazza posts or communicate with anyone about the project. You **are allowed** to make private posts on Piazza, which will be visible to the instructor and TAs, and we will be able to answer questions in that manner.

19. You are **not allowed** to use code from the internet or other sources (to be clear, including any code from OpenCores or anywhere else). The code you write must be original.

20. As a basis to this project, you can use any of your previously submitted labs (even if submitted jointly with another person), or you can make a new project. That is up to you.

21. As a first step, I suggest porting the VGA core to Altera without yet integrating it into the 68K system. This means porting the contents of vga80x40_test.vhd to display the demo screen from the DE1-SoC. In order to do this, you will need to port the font and text memories to Altera. You already have the MIF files for this. You need to figure out which types of memories to generate and how to connect them. This you should know by now, based on your previous labs.

22. Once you have the demo screen working, you will need to map at least one of the VGA core's memories (which one?) to the 68k's address space. Furthermore, you will need to map several registers to control the cursor appearance of the VGA core (see the core's documentation).

23. In the student template's "software" subdirectory, you are provided shells of the software you need to write in the files "snake.h" and "snake.c". These files contain

the game's code, but will not work until you write several functions and modify it further.

24. The "C" functions that you will need to write are contained in the section labeled "functions to implement" in "snake.c". You may want to add some more constants and/or definitions to "snake.h", as necessary.

25. Note that the function "clock()" needs to return the time in milliseconds. You **must** generate this time with the aid of interrupts, generated by a hardware timer. A resolution of 10ms is OK, i.e. a timer that causes an interrupt at a rate of 100Hz is OK. You should know how to do this by now.

## Part 4: Testing and submitting the project

26. **Your solution must print out in the hyperterminal console your name and student number when it boots up. Solutions that do not do this will not be accepted.**

27. You need to replicate the provided solution's behaviour, in particular pay attention to the following:

28. Make sure to have the demo screen pop up as the first thing that shows up when you load the SOF. Make sure the cursor is small and blinks in the middle of the screen, as in the solution.

29. Make sure the game behaves identically to the solution. Make sure the score is updated every time you eat food. Make sure the cursor is deactivated while the game is playing.

30. Make sure the "Game Over" screen has the same behaviour as the solution, namely the cursor is big, there is a delay (of 100 milliseconds) between writing each letter, and that after the Game Over screen is

written its colors change like in the solution (a color change every 300 milliseconds).

31. Submission of the code will be done via Canvas. Make sure to upload the code well before the deadline, as a ZIP file of your entire Quartus project directory (you may exclude the "db" and "incremental_db" subdirectories). Just like labs, **include a video that shows your snake game working.** Make sure to include a README.TXT file in the top level of the ZIP file that you submit which includes the following:

    - Where (which directory) is your SOF file located, and what it is called
    - Where is your project code located (Verilog and software code)
    - Where is the video located, and what is it called
    - What is the status of the project (what works, what doesn't)
    - Any additional information that would be relevant for the TA marking your project.
    - The idea is that the TA, if he or she so chooses, can compile your hardware and software project and recreate your video. Usually the TA will not do this and rather rely on inspection of your code and video and testing of the SOF, but the TA will be able to compile and recreate the SOF in case he or she has any doubts or wants to do a random spot-check of the submission.
    - The TAs may request a live demonstration of your code in which they will ask you

questions. This will be done either in person or via Zoom.

## Grading:

20%: Basic porting of the VGA core works, as verified by the VGA demo screen

15%: Integration of the VGA core into the address space in hardware and software is done correctly

5%: Generation of interrupts via hardware timer for clock generation working

20%: Software functions for snake implemented correctly

40%: Snake game works like the solution,
including Score counting and display works (5%),
Game Over screen works like the solution (10%)

Good Luck!!!!!!