CPEN412 Winter 2023 Term 2
University of British Columbia
Final Project

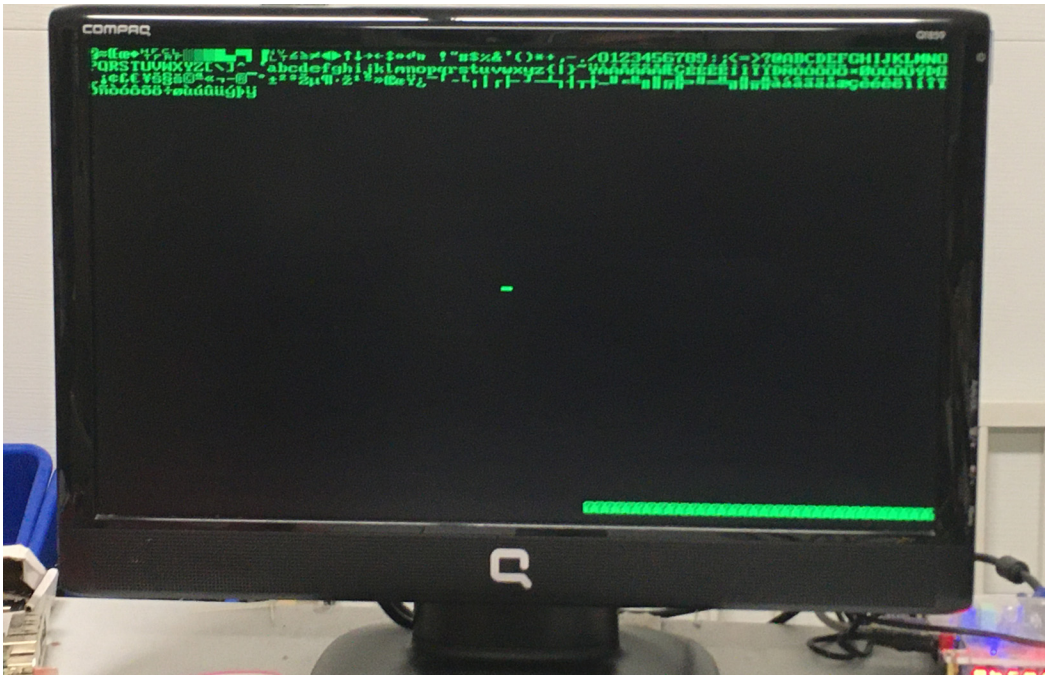
In this final project we will build the retro Tetris arcade game. This will include integrating a VGA display core into the project, mapping it to the processor's memory space, writing appropriate functions to operate it, and integrating those software functions into the game's C code, as well as modifying the code as necessary. Furthermore, the Tetris game will have a speech synthesizer component that will emulate the SPO256 speech synthesizer chip from the 1980s. The speech synthesizer will be controlled by the 68k processor.

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

1. Download the "final_project_2023_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. Connect to the DE1-SoC a VGA screen and headphones or speakers to the green audio output. You will see a demo screen, as follows:



4. Press enter to see the options in hyperterminal. To play the Tetris game press "T" in the hyperterminal. Note that pressing "H" in the hyperterminal will sound "Hello World".

5. Everyone is probably aware of how to play Tetris. In case you've been living under a rock, or maybe in case students are much younger than I thought and are not familiar with this gaming classic, see https://en.wikipedia.org/wiki/Tetris to get familiar with it, and you can play online at: https://tetris.com/play-tetris

6. The control of Tetris game pieces in our game is achieved via the hyperterminal. Pressing "w" will make the Tetris piece rotate by 90 degrees, "a" will make it go left, "s" will make it go down, "d" will make it go right. Once a game has ended, you can press "T" again to start a new game.

7. When a player completes a line, the computer will say one of the following (chosen randomly): "cool!", "awesome!", "yeah!". This is true for each line achieved, so if the

player completes two lines, two random words will be said in succession, etcetera.

8. When the game is over, the computer will say "Game Over" and display the game over screen.

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

9. Download the student template file:

   final_project_2023_sem2_student_template.zip

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

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

12. 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.

13. 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.

14. 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).

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

16. 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.

17. 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.

18. The file "vga80x40_test.vhd" would display the demo screen on a Xilinx device. While it will not compile for Altera, it could give you an idea on how to use the VGA core, in addition to its documentation of course.

Part 3: Completing the project

19. Compile and load the template project MC68K.qpf. When you do so, you should hear the following words spoken: "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, I hate COVID-19". This will repeat indefinitely.

20. In the top level file MC68K.v there are the following encrypted submodules, which are included in the project as ".qxp" (Quartus partition) files: **speech_synthesizer_wrapper**, **speech_subsystem** and **I2C_AV_Config**. You do not need to (and cannot) view or modify these

modules, but they will help you complete this project.

21. The final project is **individual**. You **are not allowed** to post public Piazza posts or communicate with anyone about the project (except, as noted earlier, to get the provided solution SOF working). You **are allowed** to make private posts on Piazza, which will be visible to the instructor and TA, and we will be able to answer questions in that manner.

22. 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.

23. Look at the file **spo256a12.pdf**. This is the datasheet of the voice synthesizer chip that is emulated by the combination of speech_subsystem and I2C_AV_Config. In particular, look at pages 7 to 18 of the datasheet which discuss the phonemes and how they are used.

24. The module speech_synthesizer_wrapper generates the phoneme codes and control signals which tell the speech synthesizer to say the words "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, I hate COVID-19". You will replace the connections to this module with connections to the 68k processor, which will run software that will provide the phoneme codes and control signals in order to say the words as is done in the solution.

25. As a start for the software project, use the IDE68K project in the subdirectory Programs/DebugMonitorCode provided in the template. You are provided shells of the software you need to write in the files "tetris.h" and "tetris.c". These files contain the game's code, but will not work until you write several functions and modify it further.

26. Note that the hardware project contains an encrypted SDRAM controller already connected and working, with the software set up to use it.

27. Look at the files "say_phoneme.c" and "say_phoneme.h", which provide functions that will be very helpful to control the speech synthesizer. They will work once you figure out a way to write the function:

```
void say_phoneme   (char phoneme_code);
```

28. In order to write the function say_phoneme, you need to figure out a way to output a phoneme code to speech_subsystem and supply the necessary control signals. Note that in order to make speech_subsystem output a phoneme, you need to do the following:

    (a) The signal **phoneme_sel** (8-bit) needs to be the phoneme code.

    (b) The signal **start_phoneme_output** needs to be pulsed for one clock cycle to initiate pronounciation of the phoneme.

    (c) Your software needs to monitor the signal **phoneme_speech_busy**. This signal will go high (and stay high) while a phoneme is being pronounced, and will go low (and stay low) once the phoneme has finished being pronounced.

    (d) Hint: if the above explanation of how to operate speech_subsystem is not enough, you can always snoop via SignalTap on the above signals and see how the speech_synthesizer_wrapper operates the module speech_subsystem.

19. Once you have figured out how to implement the function "say_phoneme", the "Hello World" functionality of the hypterterminal should

work. You should look at the code that says "Hello World", as well as the SPO256 datasheet, in order to figure out how to say words.

20. You need to then write functions that say the words "cool!", "yeah!", "awesome!" as well as "game over!", and integrate them in the appropriate code. Don't worry, the pronunciation does not need to be exactly as the solution, i.e. your phonemes could be slightly different, but the words should be intelligible.

21. As a first step regarding the VGA functionality, 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 (and probably you'd like to port to SystemVerilog from VHDL). 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. The "C" functions that you will need to write are contained in the section labeled "functions to implement" in "tetris.c". You may want to add some more constants and/or definitions to "tetris.h", as necessary.

24. 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

25. **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.**

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

27. 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.

28. Make sure the game behaves identically to the solution. Make sure the score is updated every time you complete a line. Make sure the cursor is deactivated while the game is playing.

29. Make sure the words "cool!", "yeah!", "awesome!" are said randomly when a line is completed. Make sure the game says "game over" when appropriate.

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. Bonus: Modify your project so that each Tetris piece has a different color, like the original Tetris game. For example, see here: https://tetris.com/play-tetris . With this you can earn up to 15% bonus.

32. 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 Tetris 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 (SystemVerilog and software code)
    - URL of a video in YouTube showing your project in action
    - 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:

10%: Basic porting of the VGA core works, as verified by the VGA demo screen

10%: Integration of the VGA core into the 68k address space in hardware and software is done correctly

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

15%: Integration of the audio subsystem in hardware and software is done correctly.

20%: Software VGA functions for tetris implemented correctly.

40%: Tetris game works like the solution,
including Score counting and display works (5%),
Game Over screen works like the solution (10%),
words are properly said during the game (10%)

Bonus 15%: Make each Tetris piece a different color like in the original Tetris game

Good Luck!!!!!!