CSCB58 Project File: Summer 2017  
  
Team Member A  
-------------  
First Name: Xiao Lei  
Last Name: Zhang  
Student Number: 998923820  
UofT E-mail Address: [excel.zhang@mail.utoronto.ca](mailto:excel.zhang@mail.utoronto.ca)

Team Member B  
-------------  
First Name: Won Tae  
Last Name: Jung  
Student Number: 1002433208  
UofT E-mail Address: won.jung@mail.utoronto.ca  
  
Team Member C (Optional)  
-------------  
First Name: Joseph  
Last Name: Dong  
Student Number: 100 334 9272  
UofT E-mail Address: joseph.dong@mail.utoronto.ca  
  
Project Details  
---------------  
Project Title: Road Crosser   
  
Project Description: The player controls a particle (in directions left, right, up, down) spawning from the bottom of the screen. The ultimate goal is to move the controlling particle to the top, while avoiding collisions with other particles that moves on the screen in horizontal directions.

The number of objects moving at different speeds can be set initially. There’s a score counter which depends on the y position of the player before the game ends. If time permits, we will allow the player to set the number of lives available. We may also create additional features such as an in game display of score and number of lives of the player with HEX panels on the screen if time permits.

Video URL:  
  
Code URL (please upload a copy of this file to your repository at the end of the project as well, it will  
serve as a useful resource for future development):

https://github.com/zhxl0903/CSCB58-Project

Proposal  
--------  
  
What do you plan to have completed by the end of the first lab session?:   
 - Prototype with basic features (ie: basic movement, particle generation, clock…)

- Memory and Control units should be close to finish by this time.

What do you plan to have completed by the end of the second lab session?:  
 - Complete prototype of the game with most/all features working  
   
What do you plan to have completed by the end of the third lab session?:  
 - Code is completed; final revisions and presentation preparation are finished

-Most of the final testing and touch-ups will be done during this week

What is your backup plan if things don’t work out as planned?  
 - Our backup plan would be to implement the game with just the required features   
 (particle movement, player movement, particle interaction, score), without the additional   
 features such as particle acceleration, in game score and number of lives display etc. implemented.

-We might reduce the number of particles coded depending on the time available

- We might not implement the number of lives feature depending on time available

What hardware will you need beyond the DE2 board?  
(be sure to e-mail Brian if it’s anything beyond the basics to make sure there’s enough to go around)

So far, we need none. If time allows, we will use a keyboard for movement inputs for the player.  
  
Motivations  
-----------  
How does this project relate to the material covered in CSCB58?:

* The clock is utilized to keep track of particles (which move in a pre-decided manner/pattern depending on ie: beginning location, stage level, etc.) (Clock).
* Rate dividers will be used to control the speed of the particles (3 speed types).
* Each Particle type will be controlled by a control module with an FSM.
* Particles are displayed using the VGA module to the monitor (lab 6 part 2).
* Particles’ locations and color may be saved in memory to be retrieved or updated (RAM lab6 part 1).
* Player location and color will be saved to a memory module to be retrieved or updated (RAM lab6 part1).
* We will keep track of the scores and display them to the HEX Display panels using the decoder module from our lab.
* Another decoder module will be made for displaying the score and number of lives HEX display panels on the screen. This decoder module will output the colors at each pixel of the display panel instead.
* We might allow the user to set the number of lives and number of particles of each type generated which also involve building a memory unit to store the corresponding data and controlling it with the control unit.

- A master control unit will be created to control the running of the game and the graphic updates. The observer design pattern is used whereby the master control

unit observes the data from memory and performs updates on graphics and check for winning and collisions if and only if the positions of the player or moving particle mobs change. As mentioned earlier, the player and moving particles of different speed types are also controlled by different control paths. This allows simultaneous updates to different registers in RAM to occur. (Use of Control paths; Special design hierarchy).

* A custom counter will be implemented to allow certain controlled timing. The counter will continue to output pulse once the value is reached. It is then reset by the master control whenever certain operations (collision penalty; live decrease, etc.) are performed. This allows a grace period during which effects like collisions will not affect the player’s lives (Use of counters).

Why is this project interesting or cool (for CSCB58 students, and for non CSCB58 students?):  
 - It is a simple game that utilizes CSCB58 concepts to create a familiar, straight-forward   
 Game for leisure purposes. In this respect, CSCB58 students will be able to take interest in both playing and learning from the design of this game.  
 - On the implementation of the widely renowned game concept, it is within their   
 Knowledge. Non-CSCB58 students will still be able to take interest, as it is a familiar  
 Game with straight-forward instructions, which does not require much CSCB58 knowledge to play and enjoy. Students may also become interested in this course due to the concepts encapsulated within this game (e.g. inputting using bits, VGA graphics, creating a program from modules, etc.)

Why did you personally choose this project?:

* It seemed like the most enjoyable concept to discuss and develop, while applying much CSCB58 knowledge in its implementation as specified above. Furthermore, the

Previous project, “Why did the chicken cross the road” gave us inspiration, but we felt  
 we could develop a much more interactive, intricate, well-designed program and   
 display to increase its quality. For example, we can use arrays to allow a player controlled number of enemy particles to be generated. We may also randomize the starting locations of the enemy particles if time permits.

Attributions  
------------  
Provide a complete list of any external resources your project used (attributions should also be included in your  
code).

VGA adapter  
Random number generator idea:

https://stackoverflow.com/questions/14497877/how-to-implement-a-pseudo-hardware-random-number-generator

Updates  
-------  
  
Week 1:

- Created repository (Github for project work, Google doc. for proposal) and organized method of communication (Facebook, google hangout).

- Changed project concept from a pac-man variant to “Crossing the road/Why did the chicken cross the road?” type game

- Drew diagrams, analyzed and discussed on the future implementation of the   
 program

- Development on the project began with the implementation/reuse of modules from previous labs (HEX Decoder, Rate Divider, Random Number Generator, etc.).

- Memory module has been implemented and tested. Additional testing maybe required.

- Master control and controls for cars and players were implemented

- Testing is still required on these modules

- The source code for the game is completed in 3 days

- Testing began on each component of the game

- The game can now input the lives, and number of cars of each type. Objects can now be displayed. They still can’t move due to a bug.

- Fixed numerous problems with coordinate update.

- Wiring for the modules has been tested. Errors were corrected.

- Collision detection is now being tested.

- Collision detection now works in ModelSim.

- The observer design pattern has been implemented in master control. Updates to graphics, collision detections, win checking, will only occur when the player or enemy objects’ coordinates change in the memory module.

- Additional testing needs to be made on game reset, rate dividers, and counters

Week 2:

- Rate dividers and counters have been tested individually

- Initial car x coordinates are now randomized using the 90-bit random number generator module

- Fixed a problem where player is not cleared with the right colour

- Numerous formatting issues with the code has been fixed

- Commenting have been added to most parts of the code

- Documentations have been added to each module. They are subject to further revisions as development continues.

- Work begins to create a bonus image display of the score on screen

- Python program has been written to generate the color matrices and coordinates for the different numbers to be displayed

- A read-only memory module will be implemented to store the coordinate information and the colour matrices for the various numbers

- The read only memory module for the different HEX digits is generated using Python code

- Structural problems with always blocks have been discovered. Progress has been made to redesign the structure of the master control module and memory module.

- Correct blocking or non-blocking assignments have been created for the always @(posedge clock) and always @(\*) blocks in the restructured memory module.

- Restructuring of the always blocks is complete for all modules. Registers assigned with values inside always @(\*) blocks now all have default values in every state.

- As a result of the restructuring, all major or problematic warnings from Quartus during compilation have been resolved

- Testing begins on the restructured code.

- Game can now start with newly structured code. Graphic display is functional. Additional testing is required on collision detection and winning condition checking.

- Documentations have been revised and errors have been corrected.

Week 3:

- VGA HEX Display which has been generated from Python code has been added to the main file of the project.

- Fixed a problem where x and y coordinates in the VGA HEX Display module were generated by the Python program from left to right in order

- Added offset feature to the HEX Display module to allow the HEX Display panel to be plotted anywhere on the screen as long as sufficient pixel space is available at that coordinate

- The game without the HEX Display on VGA screen has been tested in the library. All features are functioning properly. We decided to delay the implementation of the extra bonus HEX Display due to time limitations for testing. One suggestion was to add a line of cars generated based on the x coordinate of one car.

- The line of cars feature was scraped due to complexity in the actual implementation. We will need actual registers in memory to store them instead.

- All 45 cars’ y coordinates are now mixed using modulo arithmetic. Cars moving in different directions have been implemented.

- Since the basic game is now functioning based on tests results from the library session, work begins on the previously unfinished HEX VGA Display module.

- HEX VGA Score Display panels have been integrated into our program. A boundary was set to keep the player from passing over onto the score display section of the screen.

- HEX VGA Score Display panels have been tested in the library lab session. Fixed a bug where the two HEX display panels are too close to each other. Car speed and player movement speeds have been adjusted to increase the difficult of this game.

- Works begins on adding a third HEX VGA Display panel to display the number of lives on the screen when the game starts.

- The number of lives in game VGA panel has been integrated into this game. Some errors in documentation have been corrected. Codes which occupy long lines have been changed to multi-line codes.