**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 2037**  
  
**Project Description:**

Road Crosser 2037 is a 2D pixel based action game in which player must escape from evil cars chasing the player without being caught. Player controls a particle (movement can be in directions, left, right, up, and down) spawning from bottom of screen. Goal of this game is to move player particle to top of screen, while avoiding collisions with other particles that moves on screen in horizontal directions.

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

See README.md for game play instructions.

**Video URL:**

**Code URL :**

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 module and controlpath modules 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

- All features have been tested and most major problems have been resolved

- Bonus features such as VGA Score and Lives display are added if time permits  
   
**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 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 is to implement this game with just these required features   
(particle movement, player movement, particle interaction, score), without 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 the 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)

We need none. If time allows, we will use a keyboard for movement inputs for player.

**Motivations**  
-----------  
**How does this project relate to the material covered in CSCB58?**

- Clock\_50 is utilized to time update of objects in this game (which move in a pre-decided manner/pattern depending on ie: beginning location, stage level, etc.) (Clock).

- Rate dividers will be used to control speed of particles (3 speed types).

- Each Particle type will be controlled by a controlpath module with an FSM.

- Particles are displayed using the VGA module to monitor (lab 6 part 2).

- Particles’ locations and color may be saved in memory to be retrieved or updated. We will modify memory design to support simultaneous read/write to different registers in this memory from different modules. (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 scores and display them to HEX Display panels using HEX decoder module from our lab (lab1 part3).

- Another decoder module will be made for displaying score and number of lives HEX display panels on screen. This decoder module will output colors at each pixel of HEX display panel instead.

- We might allow user to set number of lives and number of particles of each type generated which also involve building a memory unit to store corresponding data and controlling it with corresponding control unit.

- A master control unit will be created to control running of this game and graphic updates. Observer design pattern is used whereby master control path observes 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, player and moving particles of different speed types are also controlled by different controlpaths. 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 a value is reached. It is then reset by 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 player’s lives (Use of custom counters).

- Cars’ initial x positions will be randomized using a pseudo-random number generation module. This generation module will generate 90 bit random numbers. This module is inspired by a post on StackOverflow.

**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. Playing this game using a DE2 board may also get students interested in learning about the hardware side of computers.

- On the implementation of the widely renowned game concept, it is within their   
knowledge to play this game given basic instructions. 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 switches, VGA graphics, creating a program in Verilog from modules, etc.)

**Why did you personally choose this project?**

- It seems like the most enjoyable concept to discuss and develop, while applying much CSCB58 concepts 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 with better graphics 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 these enemy particles if time permits. We can also create a better design hierarchy using multiple control paths, memory modules, and buffer modules. Some software design patterns, such as the observer design patterns can be used to test out high level design principles at hardware level.

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

- VGA adapter from lab 6 part 2

- Pseudo 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 and 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 future implementation of this program

- Development on this 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

- Source code for this game is completed in 3 days

- Testing began on each component of the game

- Game can now input 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 all modules has been tested. Errors were corrected

- Collision detection is now being tested

- Collision detection now works in ModelSim

- Observer design pattern has been implemented in master controlpath. Updates to graphics, collision detections, win checking, will only occur when 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 90-bit random number generator module

- Fixed a problem where player is not cleared with correct colour

- Numerous formatting issues with the code has been fixed

- Commenting has 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 score on screen

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

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

- Read only memory module for different HEX digits has been generated using Python code

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

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

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

- Due to restructuring, all major or problematic warnings from Quartus during compilation have been resolved

- Testing begins on 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 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 an offset feature to HEX Display module to allow HEX Display panel to be plotted anywhere on screen if sufficient pixel space is available at that coordinate

- Game without HEX Display on VGA screen has been tested in library. All features are functioning properly. We decided to delay implementation of 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

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

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

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

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

- 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

- The number of lives HEX display panel has been tested on the screen. All 3 panels are functioning. Some refreshing problems have been noticed

- LEDR[2:0] have been used to signal the status of different inputs. They are now functioning at the beginning of the game

- Additional revisions have been made to the commenting. Documentation still needs some code style improvements

- Player now supports looped horizontal movement in a closed space

- More typos in code documentation have been corrected

- Default output value of 0 has been added to the VGA HEX decoder module

- Fixed a bug where player moving left will loop into the VGA HEX panels if player’

y coordinate is within the y range of the VGA HEX panels