MINI PROJECT- FLAPPY BIRD

Sangnie Bhardwaj- 2015CS10291

Ankur Sharma- 2015CS50278

SPECIFICATIONS

In our project, we were required to implement a game functionally equivalent to the popular smartphone game “Flappy Bird” using an FPGA and a 16x2 character LCD, and also add 10 levels for the change of speed. Flappy Bird is a game where a ‘flying bird’ encounters obstacles in the form of vertical pipes. The objective of the game is to guide the bird through the gaps in the pipes without touching them or the ground. The game is over if either of the aforementioned events happen.

The bird changes vertical position through a user-controlled switch and continually flies to the right through a set of pipes. Collision detection is carried out using the known positions of the bird and the pipes. The level speeds can be set at the beginning of the game using 4-bit user-controlled switch inputs.

Assumptions: In our version of the game, the bird position is not horizontally fixed but moves to the right instead of the pipes moving to the left towards the bird. On reaching the end of the screen it re-enters through the left in the same vertical position as its exit. Also, unlike the original where the pipes have an upper and lower half with a random gap between them, due to LCD constraints our game has two kinds of pipes- one covering the entire top half of the LCD and the other the whole bottom half.

OVERALL APPROACH

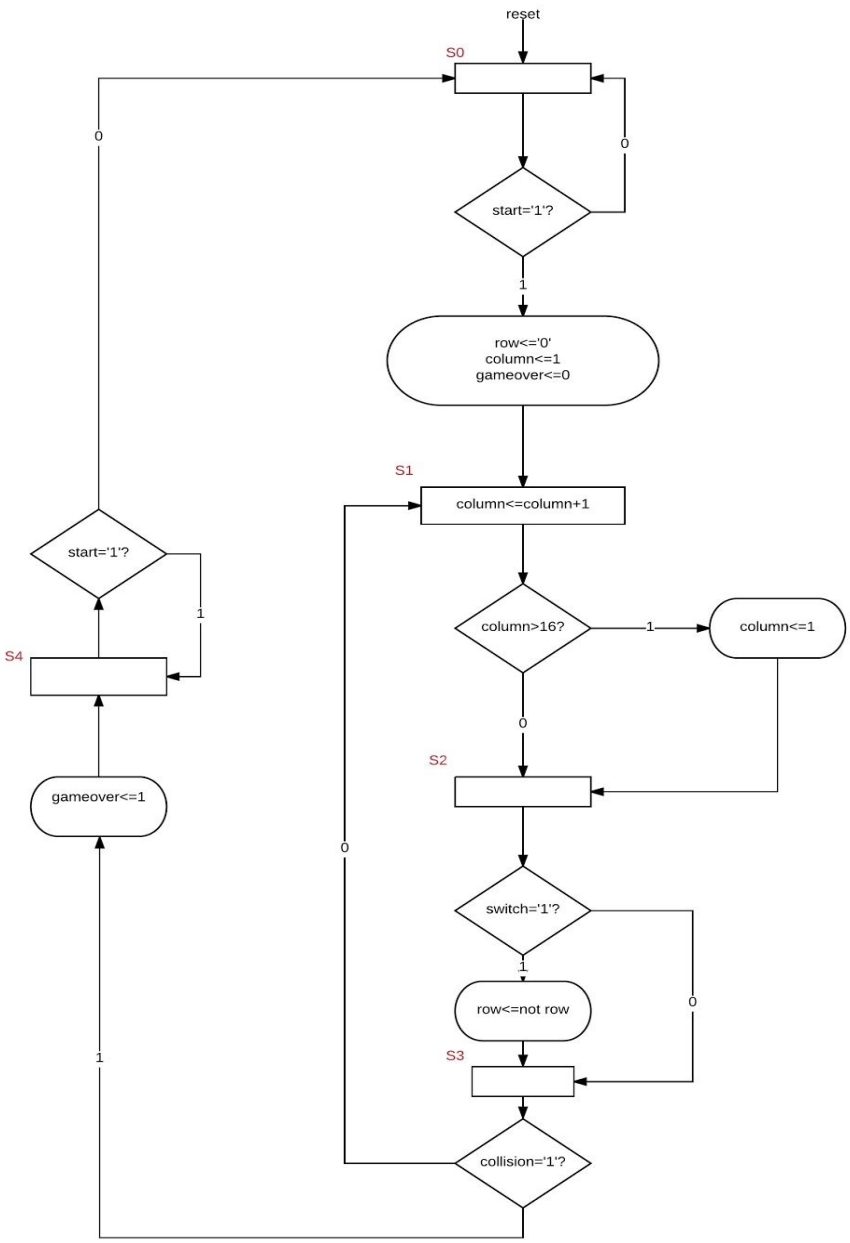
The overall approach to the game involves constantly sending bird and pipe location data, which is user-dependent, to an LCD interface which displays this on the screen. The screen is periodically refreshed according to a variable frequency governed by the speed input by the user. At each such step the bird's x coordinate is changed by selecting an appropriate position out of a set of predefined position look-up tables.

This position is selected according to a time field, which is essentially a counter taking values from 1 to 16 at the game frequency. Once the time reaches value 16, it is reset to 1, which corresponds to the bird moving back to the first column after reaching the 16^{th} column, i.e. the end of screen. The y-coordinate is changed whenever user changes switch position from 0 to 1, 0 representing upper row and 1 the lower. Together the x and y coordinates account for 32 different available positions of the bird, all of which are stored in the look-up tables mentioned before.

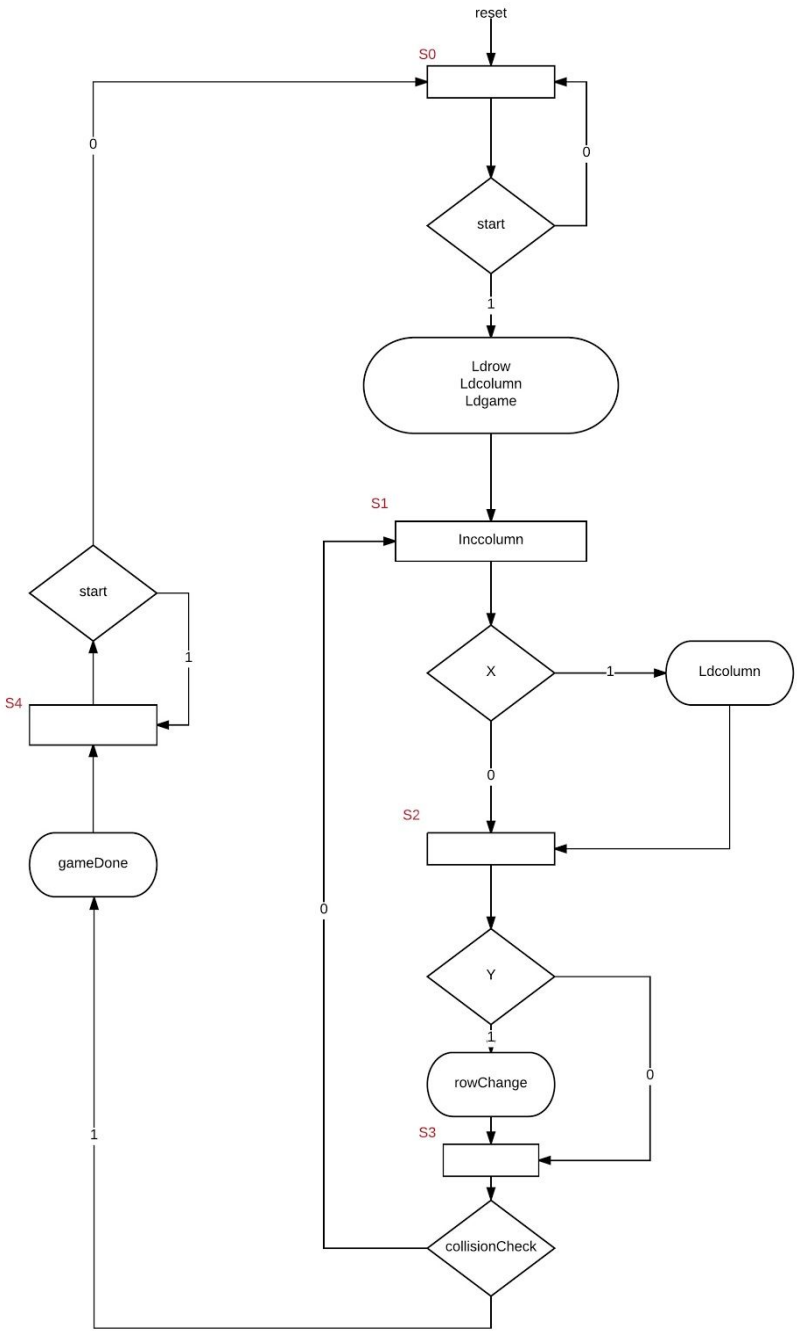
If the bird position coincides with a pipe, a collision has occurred and thus a game-over message is displayed.

Design Alternatives: A more user friendly UI-alternative to using a switch for position update was a push-button. This was not used, however, because of the debouncing in the system, which was not encountered with switches.

ASM Chart

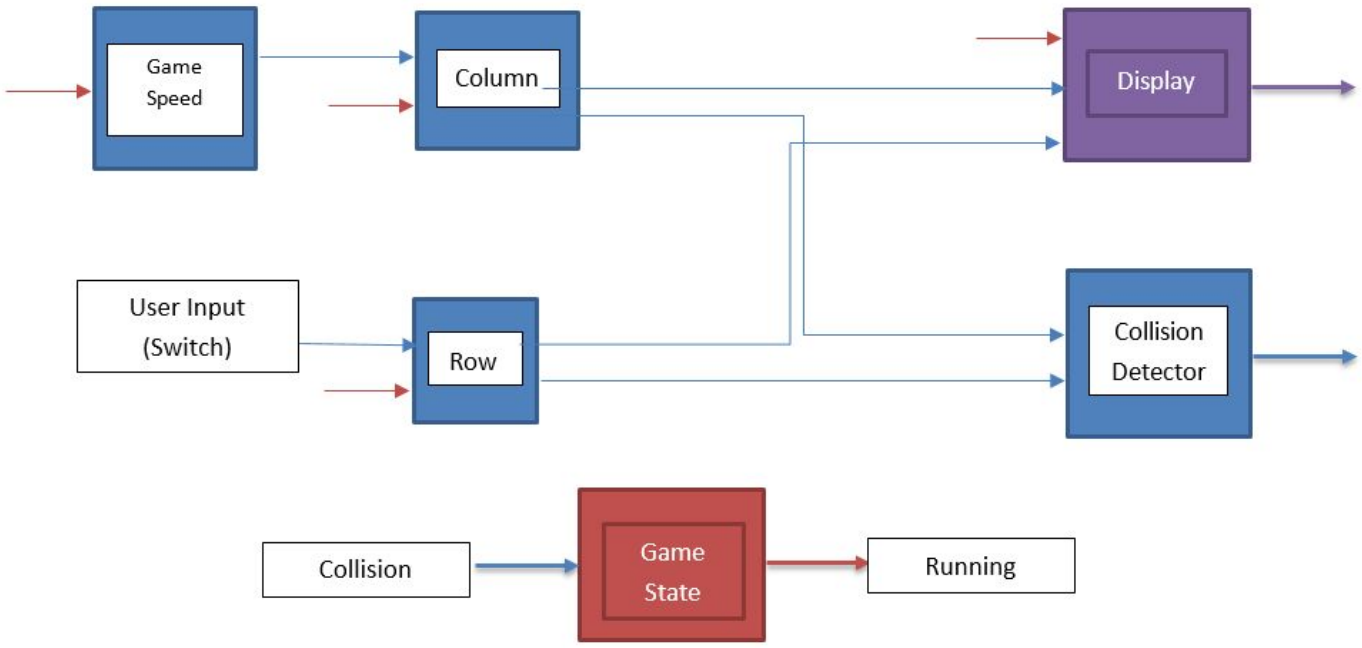


REDUCED ASM CHART SHOWING CONTROLLER

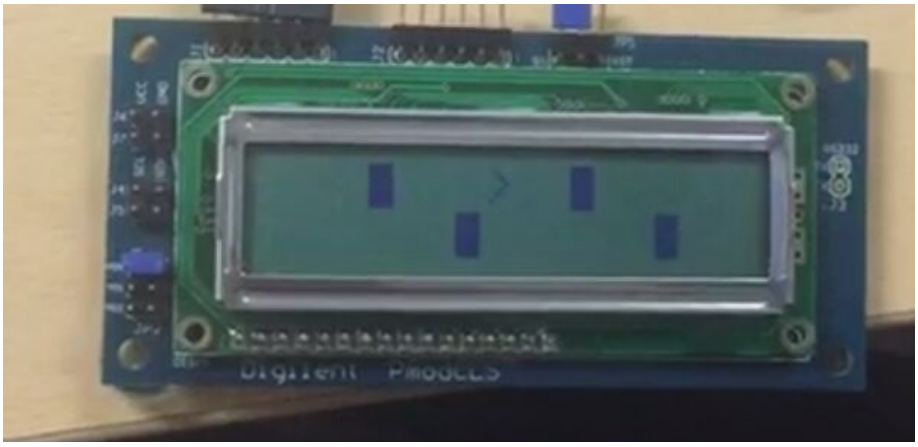


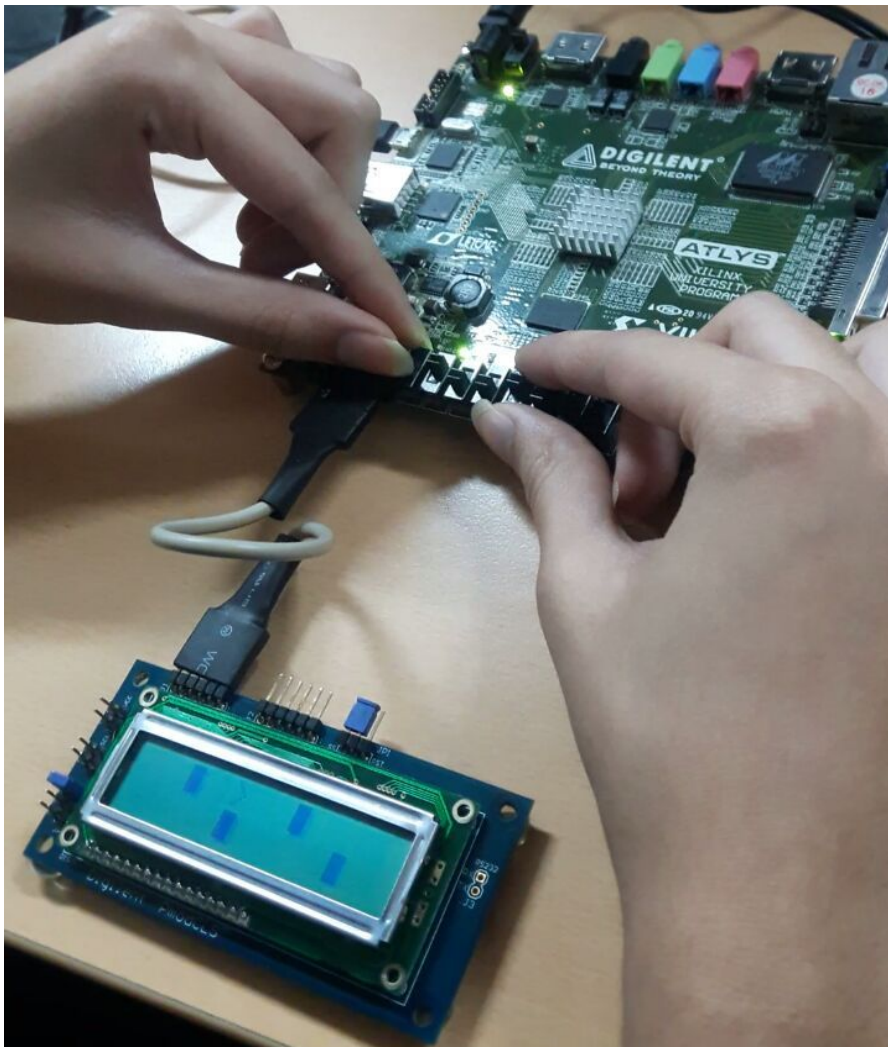
Control Signals	Meaning
Ldrow, rowChange	Used to initialize row to '0', assigns the complement value to row
Ldcolumn, Inccolumn	Used to initialize(or set) column to 1,Increment the value by 1
Ldgame, gameDone	Used to initialize gameover to 0, initialize gameover to 1
X	Checks whether column is greater than 16 or not
Y	Checks whether switch value is '1' or not
collisionCheck	Checks whether collision takes place or not
start	Used to start or stop the game
reset	Used to reset the game

Block Diagram



OUTPUT DISPLAY





IMPLEMENTATION

- **PModCLSDemo:** This is the main file of the program which contains and manages the other component instances. It also manages all the input-output of the system in the form of user-controlled switch input for bird position and speed level and the FPGA clock.
- **Master Interface and SPI interface:** These were files imported for LCD interfacing from sources referenced below. These contain the finite state machine for the LCD for outputting data to the display. These contain controls for selecting data byte-by-byte out of the command look-up entity explained next.
- **Command-Lookup:** This file contains a set of command-lookup tables, one of which is selected according to x and y coordinates of the bird. Each table contains 48 bytes of ASCII characters which determine the display of one particular block on the 16x2 screen. For our game, we chose a forward facing arrow head for the bird and a completely filled block for the pipes, 8-bit patterns for which were found in a file referenced below.
- **User Constraint File (UCF):** This contains the code for obtaining switch inputs for bird, speed, reset and clear and the FPGA clock.

Project Link(Flappy Bird):

https://drive.google.com/file/d/0BwqSn5OhABewNE1EWW9tdV8yS3M/view?usp=sharing

REFERENCES

Bird and Pipe pattern:

- http://www.pld.ttu.ee/~alsu/04_I2C_RTC.pdf

LCD interface:

- https://reference.digilentinc.com/reference/pmod/pmodcls/reference-manual
- https://reference.digilentinc.com/reference/pmod/pmodcls/start#example_projects

TESTING AND RESULTS

In this we were able to successfully create a functional flappy bird game, with more than ten different levels of speed. The game used switches to change the position of the bird as can be seen on the LCD screen. When the bird reaches the end of the screen, it reaches back to the initial column again. When the collision takes place, game over message is displayed on the LCD, denoting end of the game. Our demo showed that our game played properly with no signs of bugs, glitches or failure, while maintaining a visually appealing image throughout.