6.111 Final Project: FPGA Fruit Ninja

Lydia Sun and Nadia Salahuddin

Overview:

FPGA Fruit Ninja is a game made by Halfbrick Studios. The objective of the game is to accumulate as many points as possible by slashing through animated fruit on a screen with whatever input required (the most popular version of the game on iOS and Android use the touchscreen of the phone). In our version, the main parts that will interact with a player are a monitor and a remote. Using the remote, the player can manipulate a cursor on the monitor and play a version of the original game of Fruit Ninja.

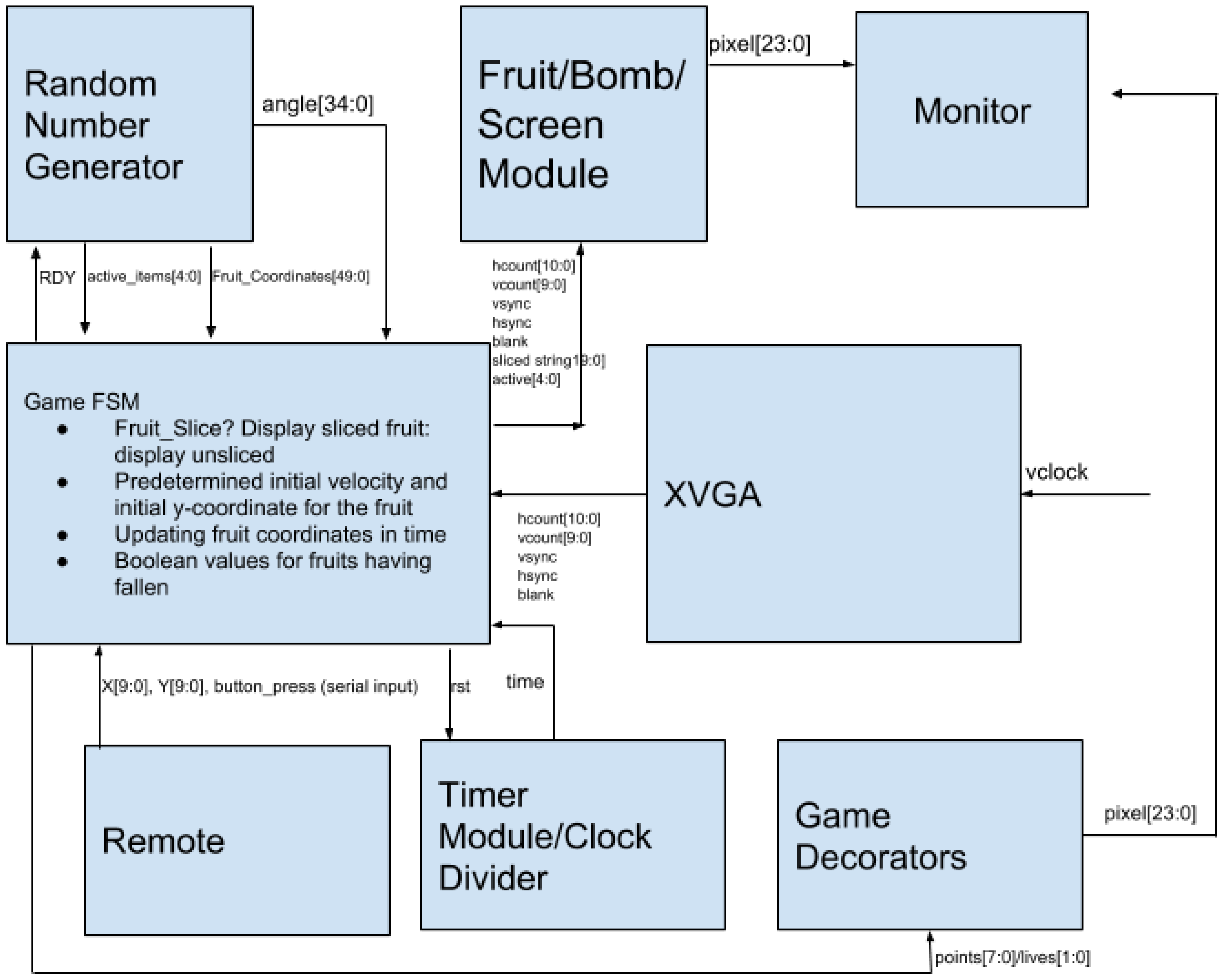
Random Number Generator
angle[34:0]
Fruit/Bomb/ Screen Module
pixel[23:0]
Monitor
RDY
active_items[4:0]
Fruit_Coordinates[49:0]
hcount[10:0]
vcount[9:0]
vsync
hsync
blank
sliced string19:0]
active[4:0]
Game FSM
Fruit_Slice? Display sliced fruit: display unsliced
Predetermined initial velocity and initial y-coordinate for the fruit
Updating fruit coordinates in time
Boolean values for fruits having fallen
XVGA
vclock
hcount[10:0]
vcount[9:0]
vsync
hsync
blank
X[9:0], Y[9:0], button_press (serial input)
rst
time
Remote
Timer Module/Clock Divider
Game Decorators
pixel[23:0]
points[7:0]/lives[1:0]

Figure 1. Block diagram of the different modules interacting in the game.

Materials:

- Nexys 4 board
- ESP32 microcontroller
- IMU MPU9255
- Button

Game States:

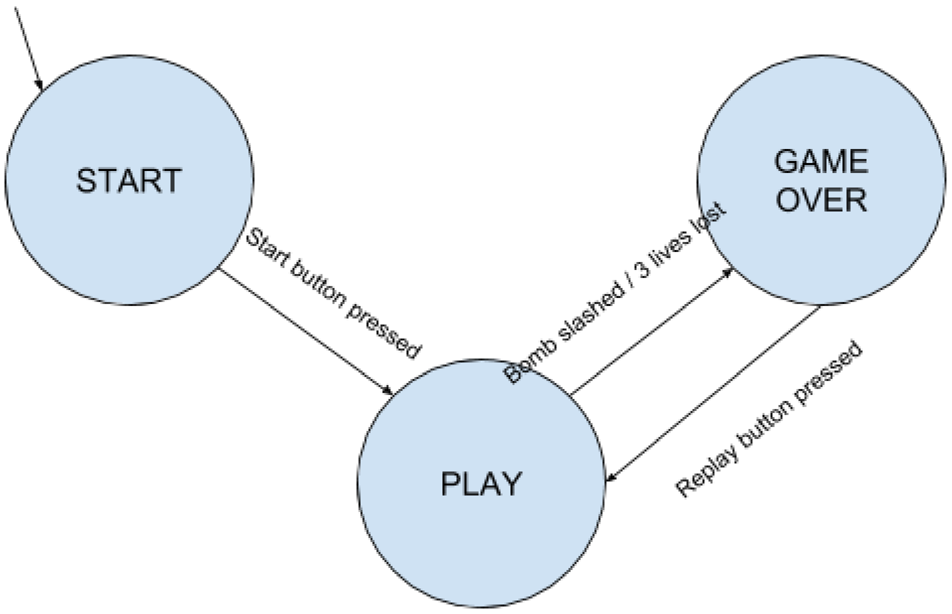

Figure 2. Game finite state machine.

The game will have 3 main states: start, play, and game over.

1. Start: This state is entered on startup. The screen displays the game title and a start button. The player can press on the start button to initiate play.
2. Play: The game takes place. If the player slashes a bomb or misses three fruits in total, the game is over. The game keeps track of how many fruit have been slashed so far, as well as how many fruit have been missed (lives lost out of three).
3. Game over: The screen displays game over, as well as the player's score at the time of the game ending. There is a button to replay.

Game Play & Rules:

During game play, fruits appear on screen as if they have been lobbed up from below. The player's goal is to slice all fruits before the fruits fall back down again and out of sight. If the player fails to slice three fruits in total, the game is over. Periodically, bombs may appear instead of fruits. If the player mistakenly slices a bomb, the game is instantly over.

Game Modules:

Random Number Generator

Inputs:

- Ready

Outputs:

- Angle (35 bits)
- Coordinates (50 bits)

This module uses a CRC output to supply random numbers for fruit launching. The launch angle should be confined to some range, which we've chosen to be 58˚ - 122˚. This is so that launch angle is not too wide and fruit do not travel too broadly horizontally. Similarly, we've constrained initial x position to the middle half of the screen, so that fruit at the edges of the screen aren't launched out of view.

Angle: Grab 6 bits from CRC output for number from 0 to 64. Add 58 to create angle in a range, so each fruit takes 7 bits to specify a launch angle.
Coordinates: Grab 9 bits for a number from 0 - 512. Add 256 to scale to the middle; each fruit's starting x is 10 bits.
Active items: random 5 bit number; each bit corresponds to an item. 1 is active.

Fruit Game

Inputs:

- X and Y coordinates of cursor (10 bits each)
- Button press (1 bit)
- Hcount, vcount, vsync, blank
- Active items (5 bits)
- Launch angle (35 bits)
- Launch coordinate (50 bits)
- Time (1 bit)

Outputs:

- Sliced (20 bits)
- Active (5 bits)

1. The module computes x and y coordinates of each fruit. On a new fruit launch cycle, assert ready to the random generator and store launch angle/coordinate of new fruits. At every time step (grabbed from the timer module), compute new x and y coordinates. If all fruit have fallen below the line, a new launch cycle begins and ready is asserted again. However, if any of these fruit are still whole, the game state machine will subtract one life.
2. The module computes whether fruit or bombs have been sliced by comparing x and y coordinates of each item to the cursor. If bomb is sliced, the game changes to the "game over" state. If fruit is sliced, the state machine will calculate the angle of the slice by drawing a line between entry and exit point., and this will determine the version of the fruit that will subsequently be displayed. The

slice value ranges from 0 to 4, denoting image to be displayed: 0 - whole fruit, 1 - horizontal slice, 2 - vertical slice, 3 - 45 degree slice, 4 -135 degree slice.

3. The module keeps track of how many fruit have been sliced so far. The state changes from “play” to “game over” if all the lives are lost or a bomb has been sliced.

Fruit & Bomb

Inputs:

- Sliced
- Active

Outputs:

- Pixel (24 bits)

Five of these modules will always be in play at any time: 4 for each fruit, and 1 for each bomb.
Sliced is a 20 bit signal, with 5 bits for every fruit. Active is a 5 bit signal denoting whether each item is active or not. If an item is active, its module is instantiated and it appears on the screen. The slice bits corresponding to each item denote which image should be displayed.

Timer

The signal from timer is used to determine time passing in order to calculate coordinates for falling fruit.

Graphics

Screens

All three game states will have one standard background image. Start and game over screens will have sprites overlaid on this image to act as buttons (start and replay). A screenshot of the original Fruit Ninja during gameplay is displayed below; our play screen will be modeled after this one:

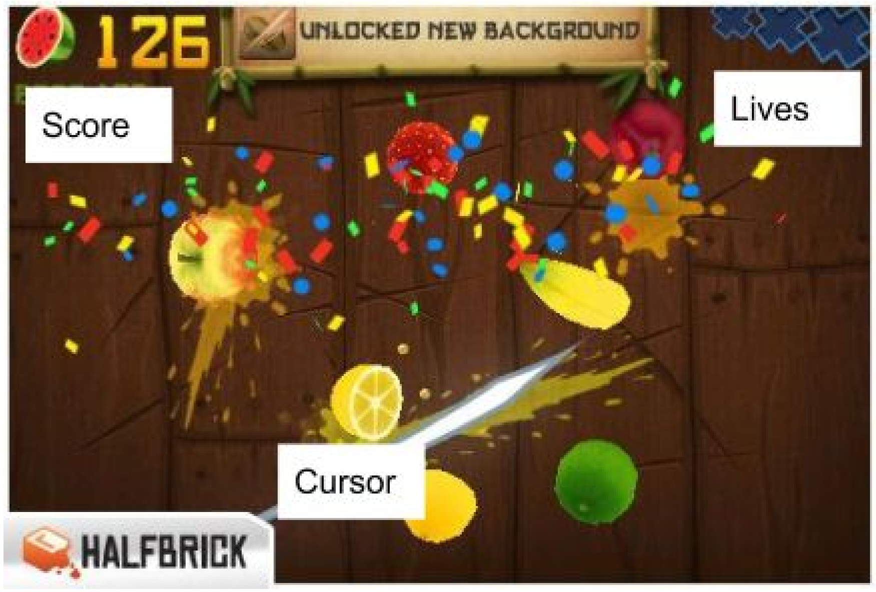

Figure 3. Snapshot of the original Fruit Ninja game being played and the various features that decorate the game.

Remote

The remote consists of an ESP32 microcontroller, a button, and an MP9255 IMU. The ESP32 reads acceleration data from the IMU and integrates it to create position coordinates for the cursor. If the button is pressed, x and y are calibrated back to the center of the screen. The remote sends information to the labkit via a serial connection, with 21 bits in total (10 bits for each coordinate and 1 bit to denote whether the button is pressed).

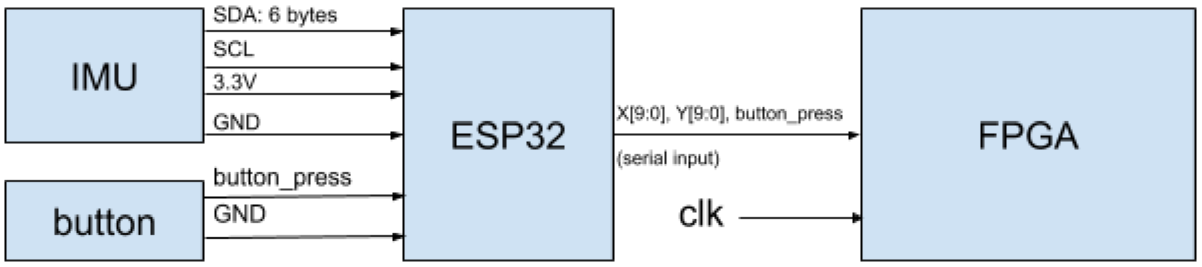

Figure 4. Block diagram of the remote unit

Potential Performance or Hardware Limitations

- How to accurately generate position from IMU data? After playing a beta version, we can determine whether a calibration feature needs to be added, or if human calibration is sufficient (players adjusting to drift).

Stretch Goals

1. Wireless remote: Change the wired serial connection of the remote to the labkit to a wireless Bluetooth connection.
2. Music: There could be music triggered by entries to different states in the game state machine. For example, sounds could be stored on the on-board memory and played at the start screen, during play time, when a fruit is slashed, when a bomb is slashed, and when the user is at the game over screen.
3. Improved graphics/mechanics: This would include adjusting the motion of fruit proportional to their real-life weight. Additional features could also include the functionality to add your own “fruit” and “fruit splitting” graphics that are more life-like.
4. Increasing levels of difficulty: as the game progresses, more fruit can be lobbed faster.
5. Bonuses: The original Fruit Ninja has bonus point systems to award the user with bonus points if they slash through multiple fruit within a short time frame or multiple fruit at once.

Timeline

Date	Goal
Nov 5	Lydia: ESP32/IMU interfacing & integrating position from acceleration Nadia: Fruit/bomb module + VGA setup
Nov 12	Lydia: Serial connection to labkit & displaying cursor on monitor Nadia: Developing fruit motion mechanics
Nov 19	Lydia & Nadia: Testing and adding in slice mechanics with button-controlled cursor, creating game interface (score, lives, background)
Nov 26 (Thanksgiving weekend)	Lydia & Nadia: Debugging and integrating
Dec 3	Lydia & Nadia: Add in real cursor, attempting stretch goals
Dec 10	Nadia: Create game screens/debugging Lydia: Integration of stretch goals/debugging