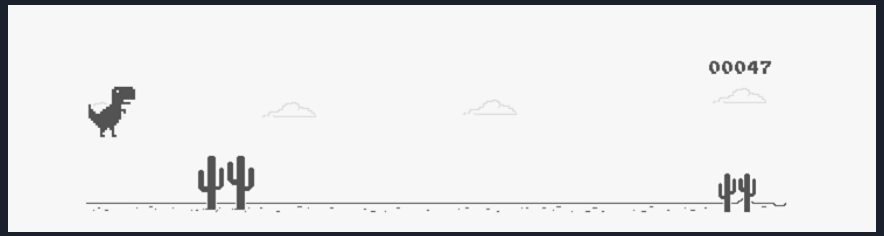
T-REX Run!
Implemented in VHDL

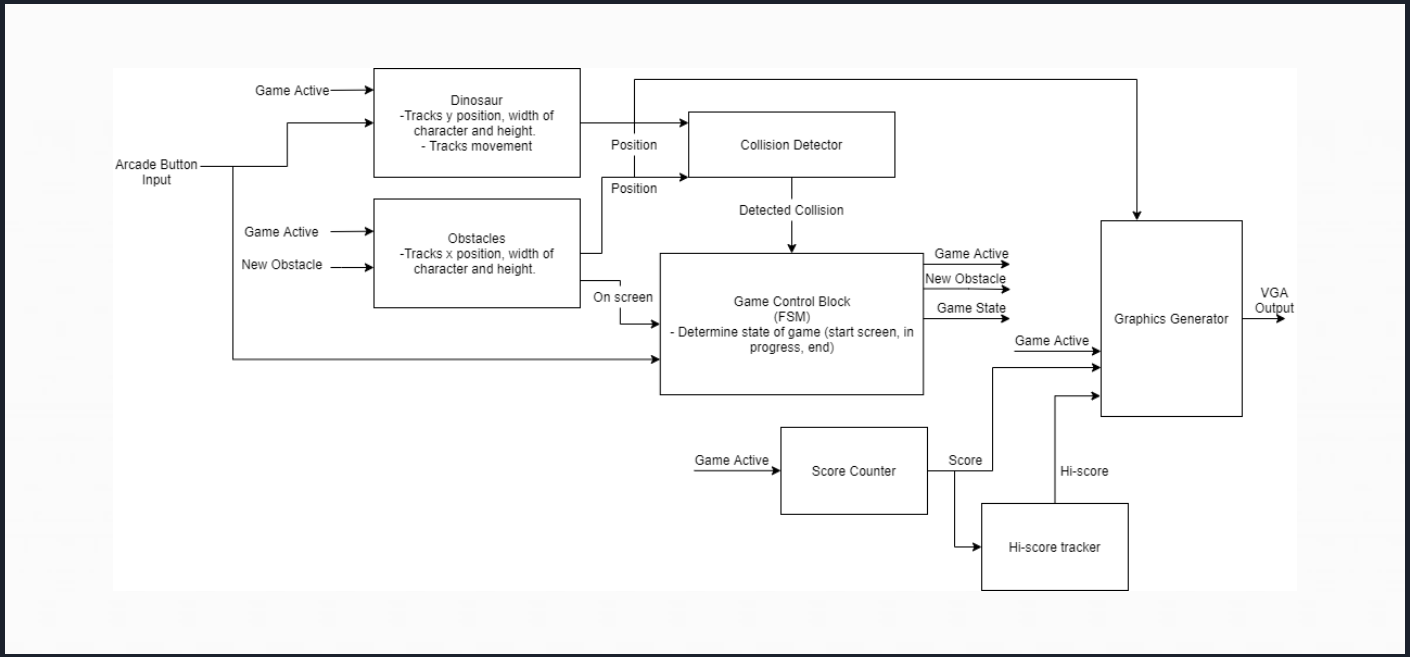
Matthew James Bellafaire
Khaled Jarrah
Conner McInnes

The Idea

- Based on google's "T-Rex Run!" game
- Game mechanics:
  - T-Rex can jump
  - Cacti moving from the right to left
  - If a cactus collides with the T-Rex it is a game over
  - Score is incremented over time the longer the player survives
  - Hiscore is tracked between gameplay sessions



The Plan


Game Active
Dinosaur
-Tracks y position, width of character and height.
- Tracks movement
Arcade Button Input
Position
Collision Detector
Position
Detected Collision
Game Active
New Obstacle
Obstacles
-Tracks x position, width of character and height.
On screen
Game Control Block
(FSM)
- Determine state of game (start screen, in progress, end)
Game Active
New Obstacle
Game State
Graphics Generator
VGA Output
Game Active
Game Active
Score Counter
Score
Hi-score
Hi-score tracker

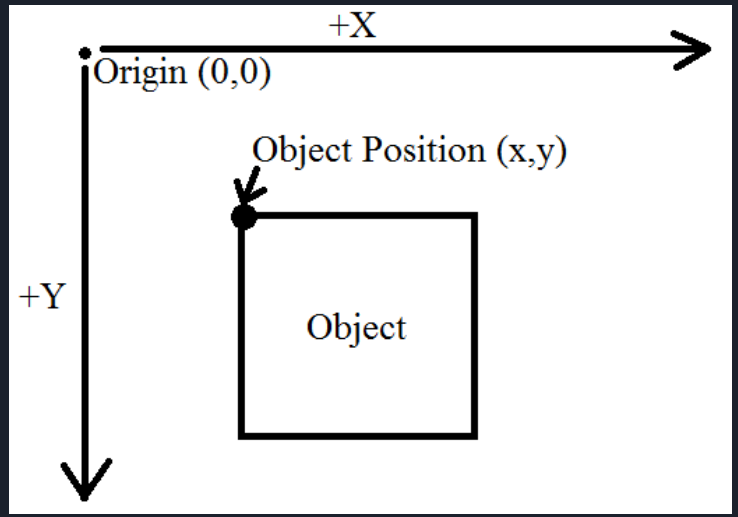
Graphics Generation

Many components are required for the proper rendering of objects on screen.

All objects generated on screen utilize a LUTs to store the how they appear on screen.

The most interesting of which is how the scoreboard is generated. It uses a parametric definition to generate the 6 displays for the BCD numbers.

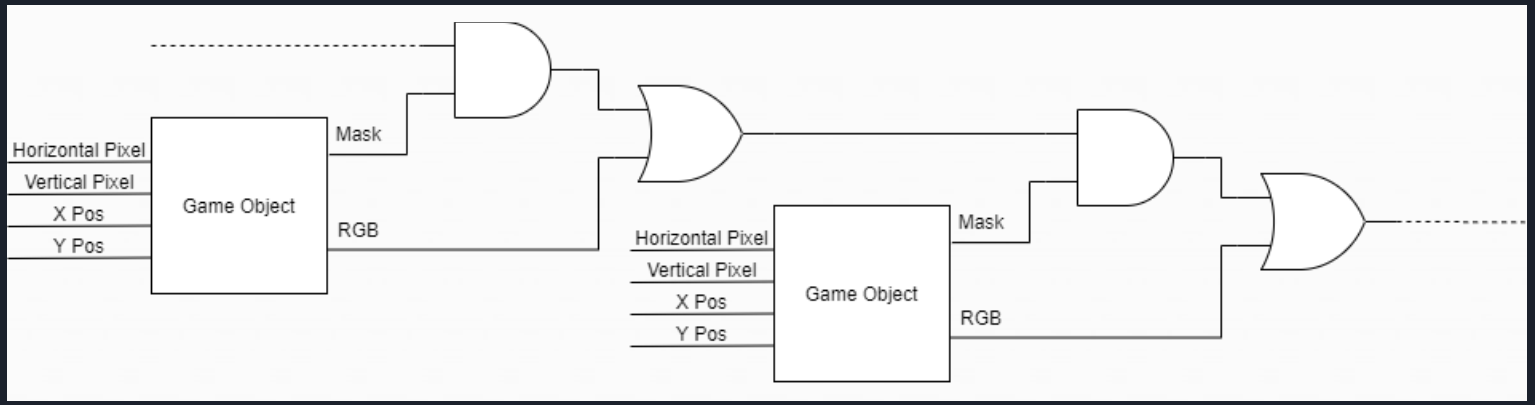
Pictured on the right is how the screen space is defined for the VGA display.



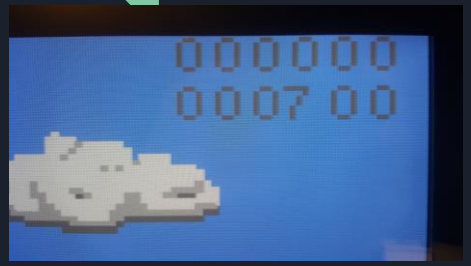
Additional Pictures of Objects

Rendering to the Display

- The display of the moving objects on screen was realized through the use of the vga_ctrl_simple.vhd file provided through Dr. Llamocca's online resources.
- All objects are rendered to the screen but are selectively made visible through the use of masks that would determine which objects would be displayed.
- Interfacing the VGA control circuit with the game objects posed some challenge.



Scoreboard



$$D_n = \frac{z \bmod 10^{n+1}}{n}$$

- The scoreboard is a parametric component which creates the required number of digits to represent a “max score” parameter.
- Each number digit is a separate component which has a fixed position on screen and takes a BCD digit as an input. The numbers for display are cropped out of the image seen on the left.
- A VHDL process was created to implement the equation on the left in order to convert the binary input from the score counters to BCD for display on screen.

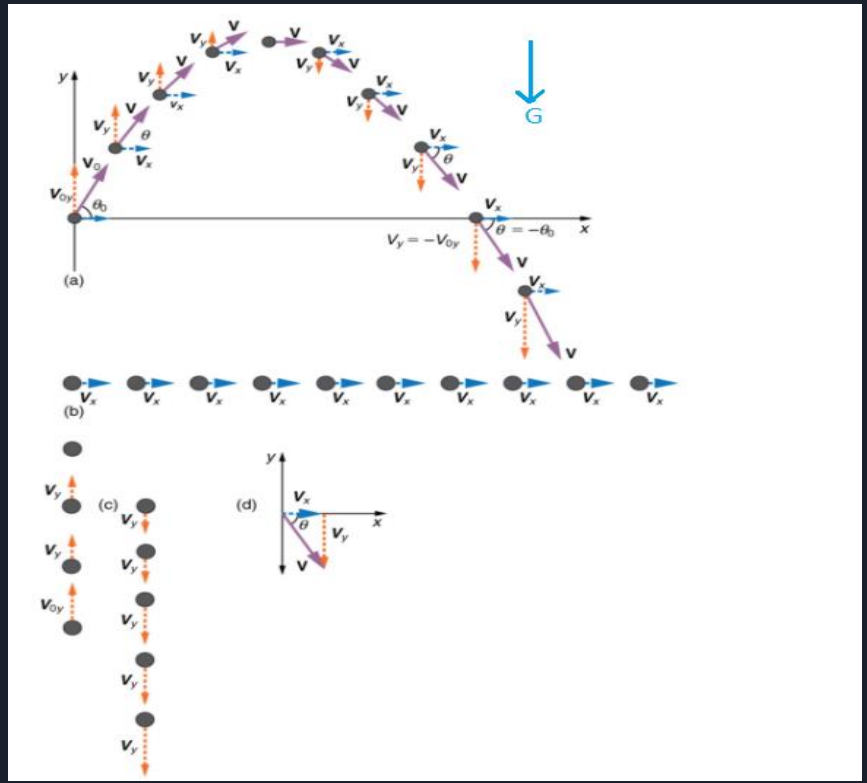
Dinosaur Jump Model

The jump motion divided along the vertical and the Horizontal axes.

The Jump height and position was set based on the net of the velocity and the gravity.

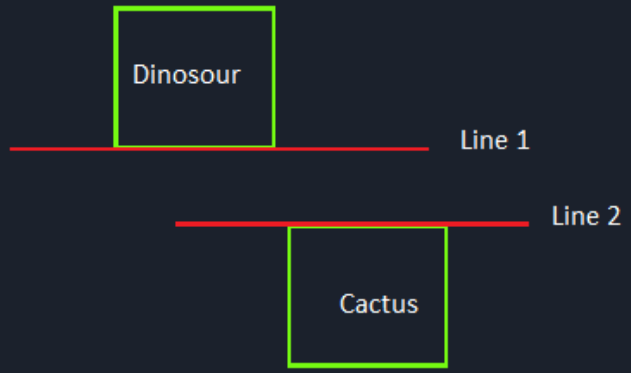
Vx is constant.

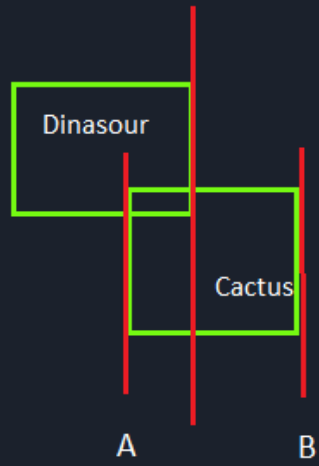
Vy is varying.

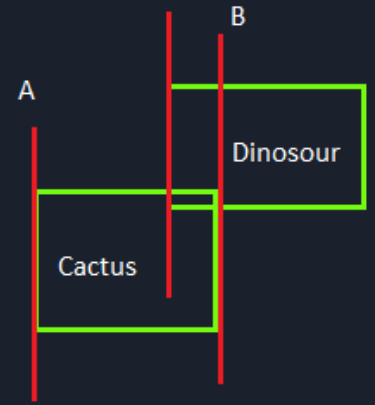


Collision model

- The collision process detects when the dinosaur hit a cactus.
- No need to check every overlapped condition.
- Hitting any cactus on the field will set the collision signal to ‘1’ and end the game







Debouncing

Debouncing was a important issue to tackle since it could affect the playability of the game.

This problem was actually able to be solved without the use of a debouncer.

In the dinosaur block, the rising edge of the jump command received from the FSM is all that was need to initiate a jump preventing any need for ensuring the signal remained high during a specified amount of time for operation.

Game Logic

Two major components make up the Game Logic.

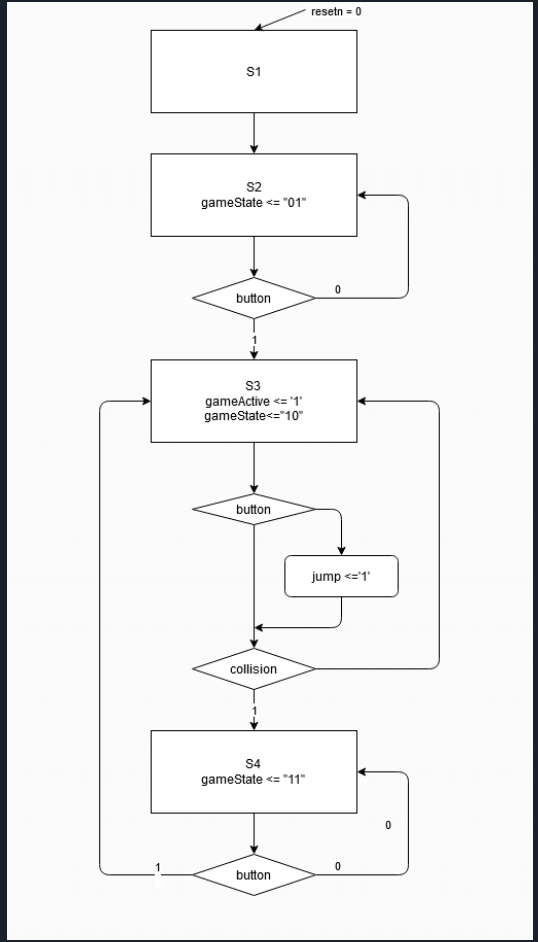
One is an FSM that controls game states and the other is a timer that controls score incrementing and obstacles.

Pictured on the left is the FSM's diagram.

The important outputs from the FSM are gameActive, gameState, and jump.

gameActive is high when the game is being played and the T-Rex can jump over obstacles

gameState defines the four states of the game:
"00" startup, "01" title, "10" gameplay, and "11" end



Game Logic (cont.)

The second part of the game logic is utilizing a counter to trigger the spawn of the obstacles after a set period of time.

When the second cactus is spawned a score signal is sent to be high which will result in 100 points to be gained

```
timer: process(clk, resetn,ga, time)
begin
obstacleEnable1<='0';
obstacleEnable2<='0';
    if resetn = '0' then
        time <= "00000000000000000000000000"; --27 zeroes
    elsif (clk'event and clk='1') then
        score<='0';
        if gA = '1' then                                --game is being played
            time <= time + 1;
            if time = 50000000 then         -- actual 50 000 000 (.5sec @50Mhz)
                obstacleEnable1<='1';                   --tells to intiate obstacle1
            elsif time = 100000000 then     -- actual 100 000 000 (1sec @50Mhz)
                obstacleEnable2<='1';                   --tells to intiate obstacle1
                score<='1';                             --increment score by 100
                time <= "00000000000000000000000000";
            end if;
        elsif gA ='0' then                              --game outside play reset time
            time <= "00000000000000000000000000";
        end if;
    end if;
end process;
```

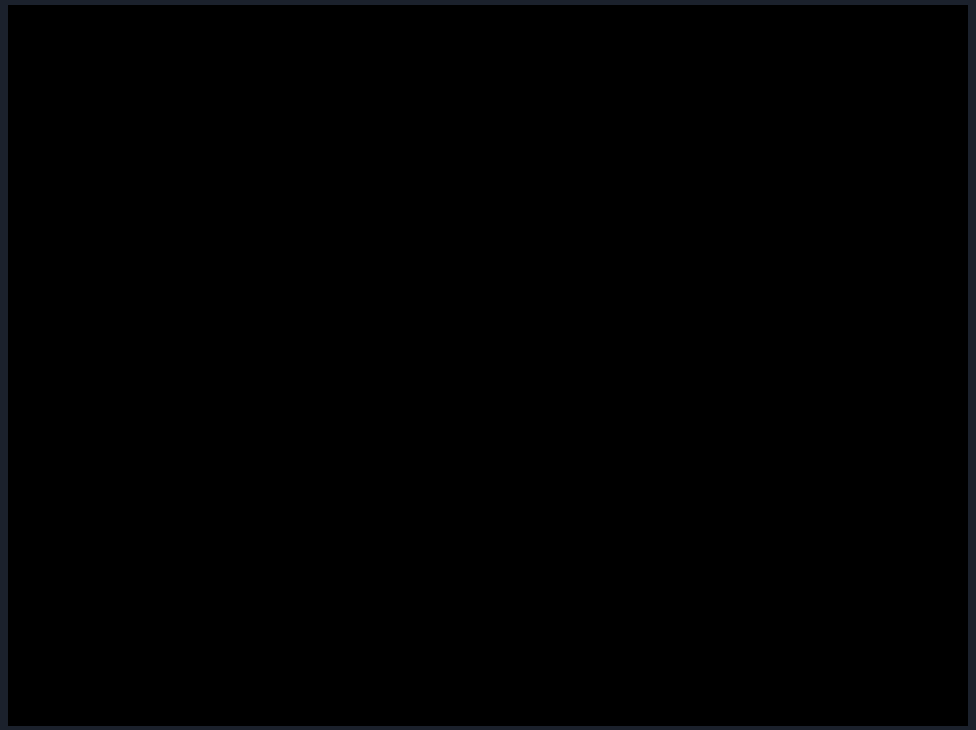
Score Tracker

This component would simply increment the score by 100 each time the score signal was sent high for a clock cycle.

It would also track the hiscore and compare if the new score achieved surpassed the old hiscore when the game ends, and if so it would assign it the new hiscore

```
process(gameActive, score, clk, resetn)
begin
if resetn = '0' then                        --reset scores
    scoreNum <= "00000000000000000";
    hiscoreNum <= "00000000000000000";
    scoreCount <= "00000000000000000";
    hiscoreCount <= "00000000000000000";
elsif (clk'event and clk='1') then          --clk pulse high
    scoreNum <= std_logic_vector(scoreCount);
    hiscoreNum <= std_logic_vector(hiscoreCount);
    if gameActive='1' then                  --game is being played
        if score='1' then                   --recieves score as high ie button is pressed
        scoreCount <= (scoreCount + 100);     --increment score count by 1
        end if;
    elsif gameActive='0' then               --game has ended or has yet to start
        if scoreCount > hiscoreCount then   --replace hiscore if score is greater
            hiscoreCount <= scoreCount;
        end if;
        scoreCount <= "00000000000000000";
    end if;
end if;
end process;
```

How did it turn out?



Unforeseen Complications

Many issues arose during the development of this project, most of which derived from attempting to interface our components with each other.

Testing the project through simulation posed difficult as well, the VGA components could not be effectively simulated.

Problem namely occured when libraries were invoked that were not available for synthesis which had to be reworked. Another when signals needed to be high long enough for them to be sensed after a new clock pulse. Finally one when calculating movement and collision of the T-Rex and Cacti.

These issues were just the majors one, numerous issues riddled the project at every stage but were overcame.