Final Project Proposal

Video Game: Astroler

Creator: Hector Soto

Class: ECE 540 Winter 2023

Description

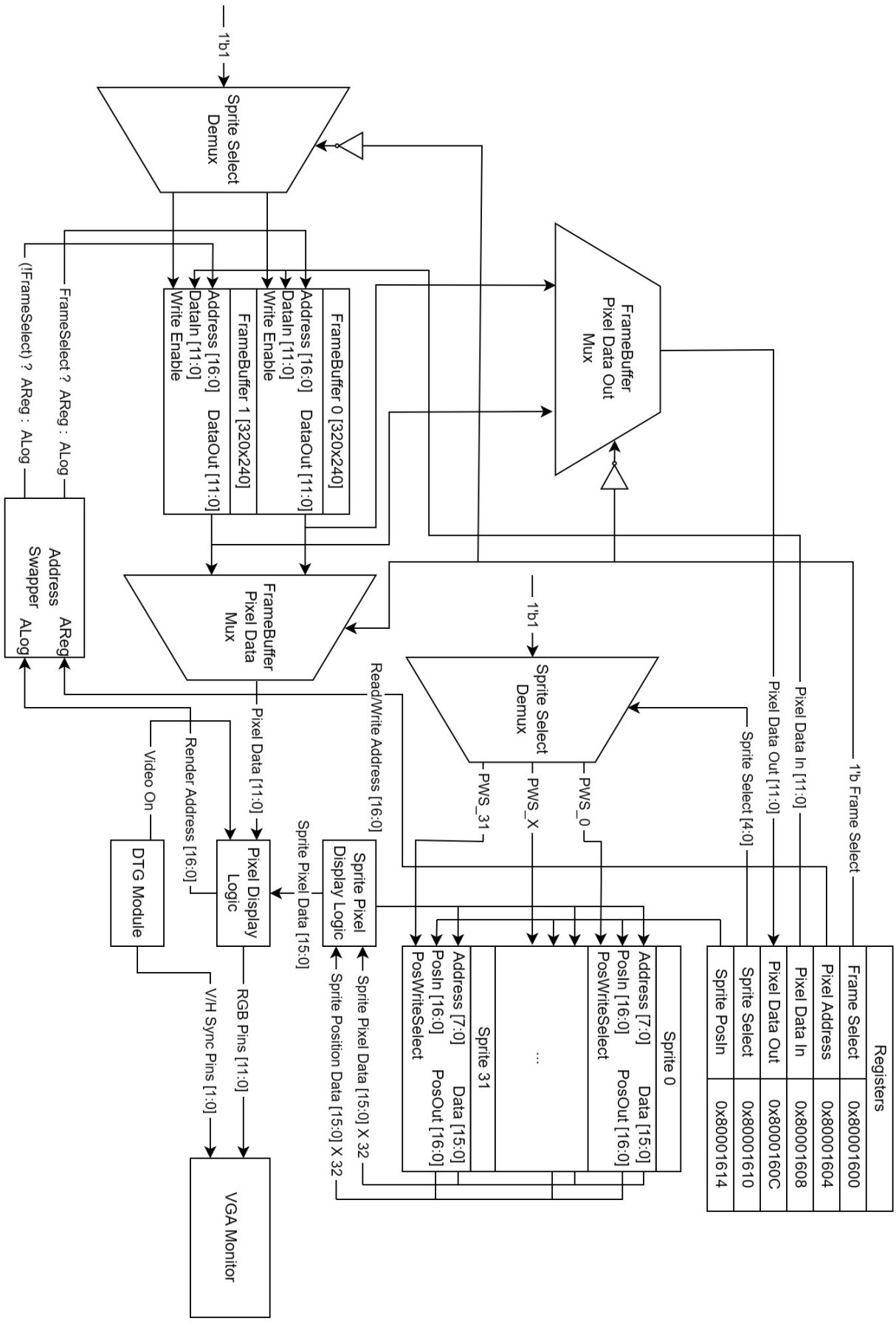
The Build

The goal of this project is to create a 2D space shooter game like Namco's 1981 Galaga that will be called "Astroler" and runs on the NexysA7-100T FPGA Development Board. Custom VGA and push-button (PBR) modules will be created to handle input, and displaying graphics. The VGA module will support two 360×240 RGB pixel frame-buffers, direct pixel reading/writing, and overlapped 16x16 multi-sprite rendering. The PBR module will support the reading of five push buttons on the board, a pseudo-random number generator, and a timer. The game itself will be programmed in RISC-V assembly. To assist in the creation of usable sprites, a sprite editor will be created using C++ to generate .coe files that can be loaded into ROM within the VGA module.

The Components

The NexysA7-100T FPGA Development Board, a VGA cable, and a basic VGA monitor will be the only physical components.

VGA Module Block Diagram



Design Approach

Hardware – VGA Module

The VGA module will have six registers, two 320x240 12-bit wide pixel frame-buffers, 32 sprite 16x16 16-bit wide pixel arrays and 17-bit wide position registers.

The 1-bit **"Frame Select" Register** controls which frame-buffer is rendered to the screen and which frame-buffer is enabled to have its pixel's written or read to. This allows the VGA module to display a frame to a screen while the next frame is being written to by software. When the Frame Select register is flipped, the next frame is shown and software can now write in the other now free frame-buffer.

The 17-bit **"Pixel Address" Register** controls what pixel in the non-selected frame-buffer is selected.

The 12-bit **"Pixel Data In" Register** controls what data is being written to the currently selected pixel for the non-selected frame-buffer.

The 12-bit **"Pixel Data Out" Register** holds the pixel data of the currently selected pixel in the non-selected frame-buffer.

The 5-bit **"Sprite Select" Register** controls which sprite's position register is selected for writing.

The 17-bit **"Sprite PosIn" Register** controls what position data is being written to the currently selected sprite.

Each sprite is a 16x16 RGBA pixel array. Each pixel is 16 bits, 4-bits for the Red, Green, Blue, and Alpha/Opacity channels. All sprites will be displayed "above" the currently select frame-buffer. Sprites are loaded from ".coe" files and can not have their pixel data overwritten.

Hardware – PBR Module

The PBR module is basically identical to the one created in Lab 4. A pseudo-random number generator will be added and its output will be available in an extra register. This random number generator can be a useful tool for the game. A timer register to keep track of how much time has passed in milliseconds will also be added for use in our game loop.

Software – Sprite Editor

The Sprite Editor will be made in C++ using the Simple DirectMedia Layer 2 (SDL2) library, this library was chosen because I've used it many times before and am used to it.

The Sprite Editor will consist of four sliders to control the RGBA levels of the 1-pixel "brush" and a 16x16 RGBA image displaying what our sprite will look like. There will also be save and export buttons to save images and export them in the ".coe" format.

Software – The Game

The game "Astroler" will consist of a player spaceship that can move in all directions in the right-half of the screen. The player ship sprite will be facing left and the background will consist of stars moving to the right to give the impression the player is flying to the left. The player will have three health points represented by three sprites in the top left of the screen. In the top right of the screen

will be the player's score. The score will go up when enemy spaceships are destroyed and go down when the player gets hit. Enemies will randomly spawn onto the screen

The background filled with flying stars will be implemented using the VGA module's frame-buffer pixel writing ability alongside the PBR module's pseudo-RNG register and timer register.

The position of sprites will of course be controlled by the VGA module's sprite position registers. If I don't want a sprite rendered to the screen, its' position can be moved off-screen so it isn't rendered.

Milestones / Goals

Week 1 (02/26 – 03/04): Read/Write Background Pixels (COMPLETED)

The goal of this week is to develop the VGA module to the point we can read/write pixel data to the frame buffers and have them appear on screen. If we fail to implement multi-sprite rendering in SystemVerilog, we can still use this feature to have multiple sprites using software.

Week 2 (03/05 – 03/11): Sprite Editor and Moving Sprite (JUST STARTING)

First goal of this week is to create a sprite editor in C++. This is to make sprite creation easy and loadable onto a ROM.

Second goal of this week is to have our VGA module render a sprite on the screen and to be able to move it using the VGA module registers accessed by our software.

Week 3 (03/12 – 03/18): Game Logic

At the start of this week our VGA+PBR modules should be complete and we should only need to program our game in RISC-V assembly. If feeling strapped for time, accomplishing this in C is also acceptable. Programming should start no later than 03/16.

Week 4 (03/19 – 03/20): Cleanup

At this point, the game should be mostly complete and only software polishing/debugging should be necessary.

Stretch Goal (Any Time): Sound Module

If there is time, I could implement a sound module that outputs a digital square signal at different frequencies based on a register written to by software. This would allow basic sound effects to be played.