ECE 385
Fall 2020
Final Project

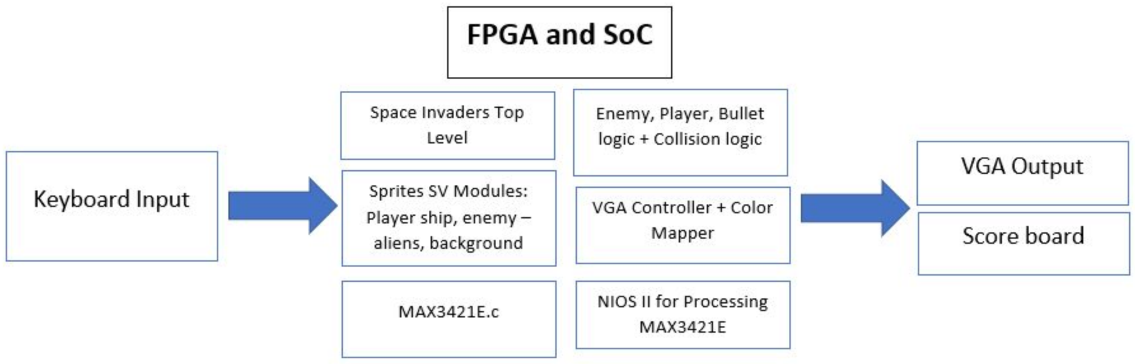
Final Project: Space Invaders on the DE-10 LITE

Putra Firmansyah (pdf2) / Kanin Tangchartsiri (kanint2)
Section ABA / 12-11-2020
TA: Andrew Gacek

Introduction:

For our final project we have implemented the classic game of space invaders which consists of a few different elements. Namely the user will control the player which is able to shoot bullets or lasers at the invading alien enemies. Once the bullet or laser hits, this will cause the enemy to die which then allows us to increment our score. The objective for the player is to obtain the highest score possible and wipe out all of the enemies before they reach the bottom of the screen. The enemies will constantly be moving laterally then followed by vertically in essentially row major order.

Written Description and Diagram:


FPGA and SoC
Keyboard Input
Space Invaders Top Level
Sprites SV Modules: Player ship, enemy – aliens, background
MAX3421E.c
Enemy, Player, Bullet logic + Collision logic
VGA Controller + Color Mapper
NIOS II for Processing MAX3421E
VGA Output
Score board

Overall this is the diagram for the set up of our system. The main inputs that are taken are the Keyboard Inputs themselves which is used to control the player sprite. The keyboard is also the signal that we use to transition from start game to the play game stage. The other input are the push buttons on the screen which are used to reset the game.

Within the FPGA and SoC we create VGA modules to output the game to the screen as well as color mappers to choose the color. The sprites themselves are instantiated as RAM modules however in actual fact given the way we use them they essentially become ROM files. This information is then pulled from the sprite right to the color mapper to be mapped on to the VGA output. The sprites themselves are for namely the background, player, and enemies. The bullets themselves are created arithmetically within the color mapper by using a single pixel's X and Y.

The project itself is separated into two main areas of both hardware and software. The software side utilizes the NIOS II processor as well as the MAX3421E which allows us to utilize the keyboard input through the DE10-Shield. This allows us to utilize the "A" and "D" key to control the player's movement. The values that represent these two keys are four and seven respectively. For the shoot functionality of our design, we have allocated the space button to be the input for this function. The keycode value for this is the decimal value 44. This is then controlled by the MAX3421E.c which then outputs this to hardware allowing us to utilize the values from the keyboard using hardware controlled modules in the SV files.

The bullet and player collision logic is created using SystemVerilog. They take in multiple different inputs and check whether or not they are out of bounds or at the same position as each other which would indicate that a collision has occurred. This allows us to track the progress of the game and destroy multiple different enemies accordingly.

The sprites themselves are created using an external program referenced from the 385 course page created by Atrifex[1]. These allow us to convert from images that are PNGs into sprites that allow us to index these onto the color mapper. In terms of displaying the sprites to the VGA, we have utilized a similar method to Will Green's Project F[2]. This is especially since the method of utilizing two coordinate systems to print out the sprites pixel by pixel.

The C functions implemented here are all used to implement the logic to write/read to the MAX3421E through the SPI which allows us to use the usb keyboard correctly. It is also important to note that in all of these functions, we can only call the alt_avalon_spi_command() function once as we would like to keep it in full-duplex mode. We can do this by either using the merge flag or a single function but we found it more convenient to do it by one single function call. To do all of them in one single function, we had to concatenate the address of the registers where we would like to write or read to, to a single array and the alt_avalon_spi_command() function should do it automatically.

SPI_wr

- This function is used to write a single byte to the MAX3421E through the SPI while also reading the status register. Reading the status register helps to see if there are any errors and helps to debug as we can assess that we are in the desired status/state.

MAXreg_wr

- This function is used to write a register to the MAX3421E through the SPI. In this function, we had to write the value, which is input into the function, and reg+2 into the alt_avalon_spi_command() write address and specify that we are writing 2 bytes in order for the function call to work properly.

MAXbytes_wr

- This function writes data and reg + 2 via SPI and then returns a pointer to the memory address of the point that was last written onto the MAX3421E. In this function, we also had to input the nbytes of data and the address of reg + 2 into the alt_avalon_spi_command() so that it would run properly and just like the previous function, we did this by concatenating them into a single array and adjusting the size of the bytes to be written accordingly.

MAXreg_rd

- This function reads a register from the MAX3421E via SPI. To do this, we write the register via SPI using the alt_avalon_spi_command() inputting the address of the reg variable into the function with a write byte size of 1 and read using a val variable and input its address into alt_avalon_spi_command() with a read byte size of 1.

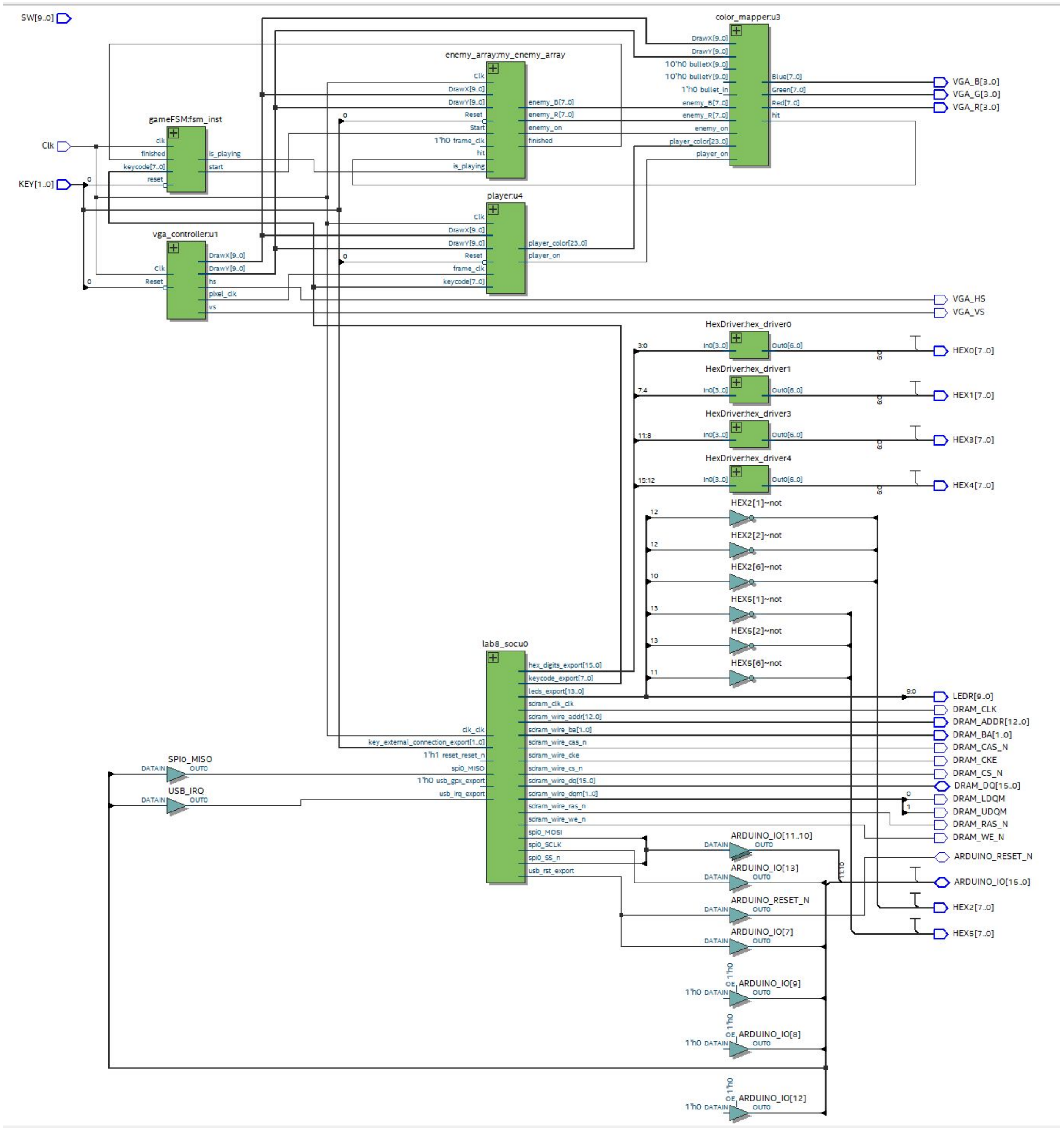
MAXbytes_rd

- This is a multiple byte write that allows us to write in registers onto the MAX3421E through the SPI. In order to do this, we input the reg address into the

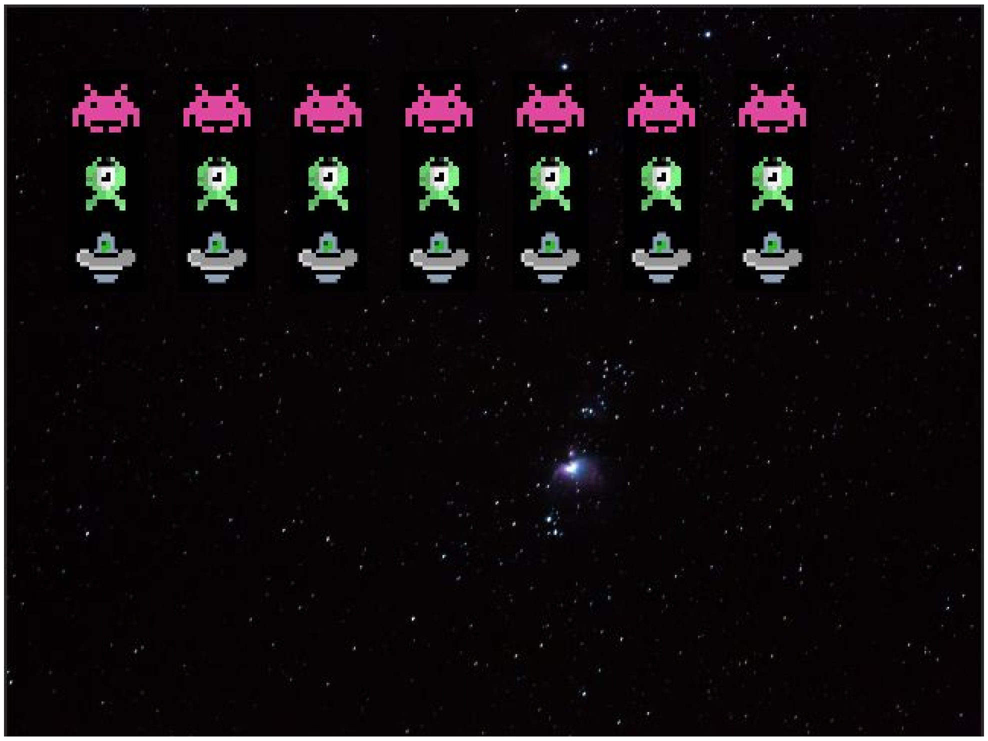
[1] https://github.com/Atrifex/ECE385-HelperTools

[2] https://projectf.io/posts/hardware-sprites/#:~:text=Hardware%20sprites%20use%20dedicated%20logic,or%20a%20few%20huge%20ones

alt_avalon_spi_command() function as the write address variable with a byte size of one, then we also input the starting address of the data array and specify the byte size of nbytes.

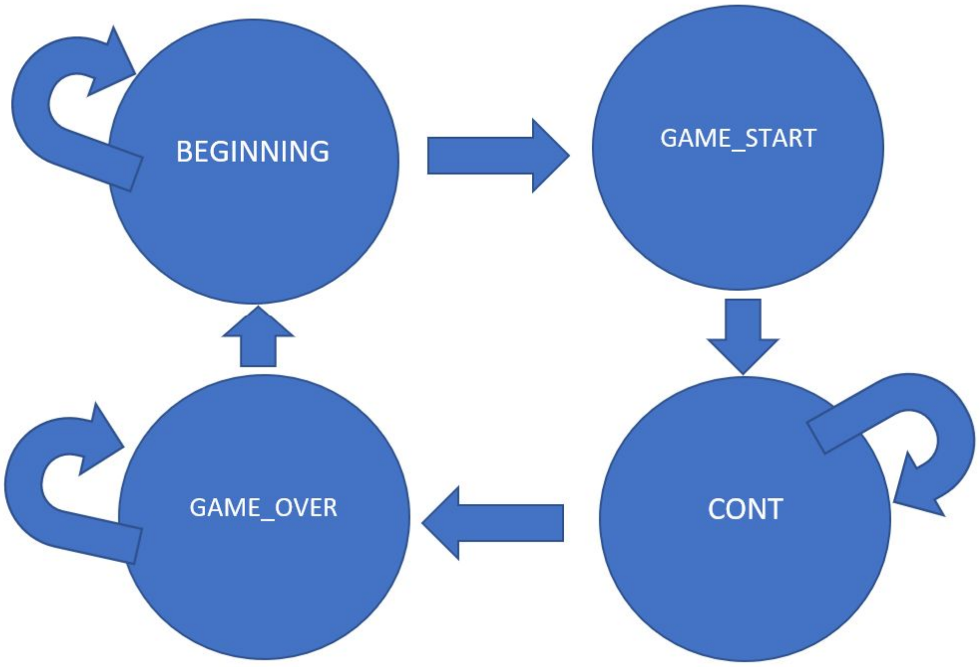


Overall Top-Level Design



Output of the enemies

State Diagram:



State Diagram of the Game

Module Descriptions:
Module: lab8
Input: Clk, [1: 0]   KEY, [9: 0]   SW,
Output: [9: 0]   LEDR, [7: 0]   HEX0, [7: 0]   HEX1, [7: 0]   HEX2, [7: 0]   HEX3, [7: 0] HEX4, [7: 0]   HEX5, DRAM_CLK, DRAM_CKE, [12: 0]   DRAM_ADDR, [1: 0]   DRAM_BA, DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3: 0]   VGA_R, [3: 0]   VGA_G, [3: 0]   VGA_B
Inout: [15: 0]   DRAM_DQ, ARDUINO_RESET_N, [15: 0]   ARDUINO_IO
Description: This module contains the top level architecture to link all of the different RAM modules and logic controllers.
Purpose: This is to connect the SoC and the different modules together to allow the logic to come together.

Module: gameFSM
Input: logic reset, input logic clk, input logic [7:0] keycode, input logic finished,
output: logic start, is_playing, is_finished
Description: This module is used to control the state of the game, allowing us to transition between the different states.
Purpose: This is to allow us to transition the different scenes on screen which means that we can output different screens accordingly.

Module: enemy_easy
Input: logic Reset, frame_clk, Clk, delete_enemies, hit, is_playing, input   logic enemy_direction_X, logic enemy_direction_Y, logic [9:0] enemy_initial_x, enemy_initial_y, logic [9:0] DrawX, DrawY, logic start,
Output: logic enemy_on, logic [7:0] enemy_R, enemy_G, enemy_B
Description: This is the easy enemy array which allows us to output the enemies that are easiest to defeat.
Purpose: This allows us to perform logical operations with the easy enemies and allows us to control the difficulty of the enemy.

Module: enemy_medium
input   logic Reset, frame_clk, Clk, delete_enemies, hit, is_playing, input   logic enemy_direction_X, input   logic enemy_direction_Y, input   logic [9:0] enemy_initial_x, enemy_initial_y, input   logic [9:0] DrawX, DrawY, input   logic start,
output logic enemy_on, output logic [7:0] enemy_R, enemy_G, enemy_B
Description: This is the medium enemy array which allows us to output the enemies that are moderately difficult to defeat.
Purpose: This allows us to perform logical operations with the medium enemies and allows us to control the difficulty of the enemy.

Module enemy_hard
Input   logic Reset, frame_clk, Clk, delete_enemies, hit, is_playing, input   logic enemy_direction_X, input   logic enemy_direction_Y, logic [9:0] enemy_initial_x, enemy_initial_y, logic [9:0] DrawX, DrawY, logic start,
Output: logic enemy_on, output logic [7:0] enemy_R, enemy_G, enemy_B
Description: This is the advanced enemy array which allows us to output the enemies that are much harder to defeat.
Purpose: This allows us to vary the difficulty of different enemies which means that we can make the game much more interesting.

Module: enemy_array
Input: logic   Clk , frame_clk, Reset, Start, hit, is_playing, input   logic   [9:0] DrawX, DrawY,
Output: logic   enemy_on, finished, output logic   [7:0] enemy_R, enemy_G, enemy_B
Description: The enemy array is used to store multiple enemies and allow us to show different enemies at once.
Purpose: This allows us to display the enemies as well as control which enemies are getting and whether or not to remove some enemies going forward.

Module: color_mapper
Input: logic [9:0] DrawX, DrawY, input logic bullet_in, logic [9:0] bulletX, bulletY, logic [23:0] player_color, logic [7:0] enemy_R, enemy_G, enemy_B, logic player_on, logic enemy_on, logic [7:0] bg_R, bg_G, bg_B
Output logic [7:0] Red, Green, Blue, logic hit
Description: This is the module that helps us decide which color to output to the screen.
Purpose: This module is used to indicate which color to output to the pixel accordingly while we have inputs from the enemy, player, bullet and background and we choose between these accordingly.

Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This is the module that converts our 4 bit binary input into a Hexadecimal.
Purpose: We utilize this section to help display the values as a hexadecimal onto the output LED’s of the FPGA. The DE-10 Lite has a limited amount of displays and so this is the way we have to display our values.

Module: vga_controller
Inputs: Clk, Reset,
Output: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, DrawY

Description: Contains logic for vga_controller to display appropriate data to the screen.
Purpose: The purpose of this module is so that it controls what and how the electron gun paints the screen. The outputs DrawX and DrawY will be output from this module and will be input to the color_mapper module.

Module: player
Input: logic Reset, frame_clk, Clk, input logic [7:0] keycode, input logic [9:0] DrawX, DrawY,
Output logic player_on, output logic [23:0] player_color
Description: This is the player module that instantiates the player RAM and performs the necessary logic to output the player onto the screen.
Purpose: This allows the user to control the motion of the player as well as the output of the sprite to the screen.

Module: alien_shipRAM, alien_RAM, backgroundRAM, pink_invaderRAM, player_spriteRAM
Input: [4:0] data_In, input [18:0] write_address, read_address, input we, Clk,
Output: logic [23:0] data_Out
Description: These are the RAM sprites that are instantiated and pulls from the TXT files to get the data for outputting onto the color mapper.
Purpose: These are used to send the information from the formation of sprites which means that we are allowed to display these on the VGA.

Design Resources and Statistics:

LUT	3252
DSP	0
Memory (BRAM)	11,932
Flip-Flop	2466
Frequency	134.99 MHz
Static Power	96.18 mW
Dynamic Power	0.68 mW
Total Power	106.18 mW

Conclusion:

Overall for this lab there were a few things that we can improve upon. Namely we should have explored a different way of portraying the sprites as they overall did not go the way that it was supposed to. This was certainly a learning experience, especially with going through the design process but a key takeaway that we need to do is to make sure that our unit tests are much more robust. Overall the ideas and construct of the program as a whole were relatively robust and thus the project in it’s ideation was a success.