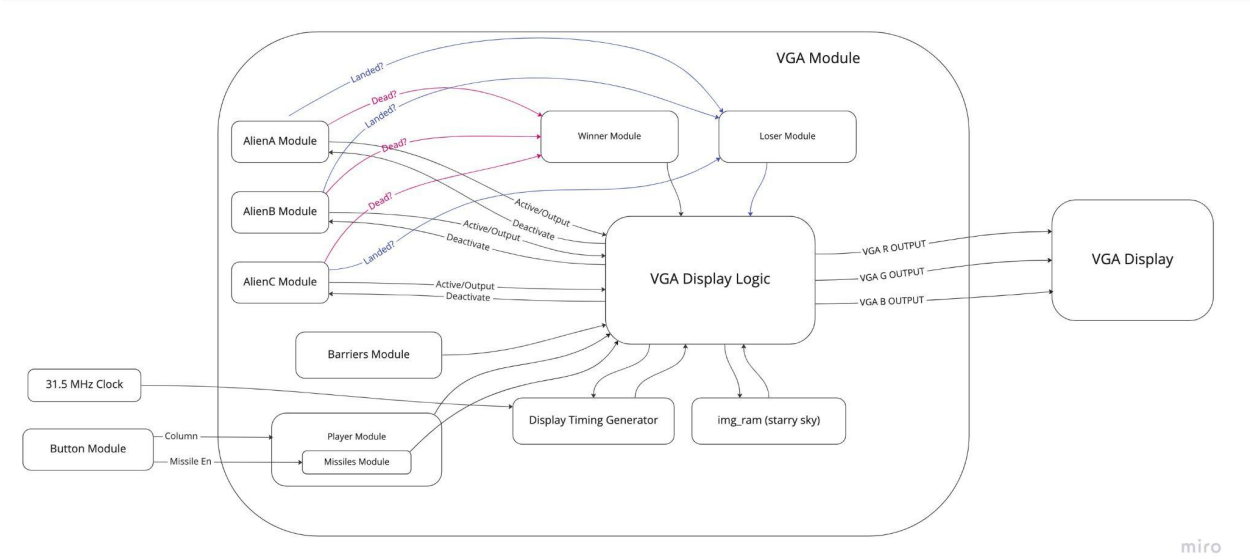
Alexander Maso
ECE 540 - Winter 2023

Space Invaders

Final Project Report

Rutuja and I have written separate Final Reports for our project since our deliverables were separate from each other's. This was the best way for us to work on a group project after we had quite a few challenges coordinating our joint work on the labs. Since Rutuja has a lot more experience with android and bluetooth connections, she was able to create a controller for our game on her own (as well as how to connect and send data to the RVfpga board) while I was able to work on the game mechanics itself. Once we both had completed our separate pieces we joined them together to create a functioning game and wireless controller.

General Design: The fpga board was used to control all of the image data as well as most of the functionality of the game. All sprites were generated through the manipulation of vga output to control individual pixels (or with a coe file and block memory generator for the background) and each sprite had their own module (each group of 5 Alien was actually contained within one module) to hold position and movement data. Movement of the "Space Invaders" as well as the missiles was implemented through the use of a Finite State Machine and a counter. Each of the modules also was fed the pixel row and pixel column data that came from the Display Timing Generator and these values helped the FSM within each module determine when to change from moving the sprites left to moving them right. Each time the counter reached its maximum value, depending on the location of the sprite relative to the pixel row and column, the position of the sprite was updated accordingly. All of this logic was implemented through "hardware" and was all written in SystemVerilog. Vivado was used to synthesize the whole design and VSCode was used to load the bit stream onto the board itself. Assembly code (also loaded onto the RVfpga board through VSCode) was used to detect individual button presses and update the corresponding registers. I have included a block diagram to show how all of the different modules connect to each other.



Design Challenges: Initially I ran into some difficulty with my design because I had far too much logic in the VGA module itself. Moving the logic for the sprites into their own modules and using the VGA module to detect which to display through the use of active flags made things much simpler. Each module was then outfitted with its own FSM that implemented motion and flags to indicate if one of the aliens had been killed or if they had reached the bottom of the display. One of the biggest design challenges I ran into after this was the missile logic itself. My design required the missile fire to be enabled through the Assembly code itself (reading a button push and updating the corresponding register), but (since the motion of the missile was controlled on the hardware side) I needed a second register to turn the missile “off” if it reached the top of the screen or if it hit an Alien. I also did not want there to be a restriction on the number of missiles that could be fired. Ultimately I was able to implement this through the use of two eight bit registers that were ‘xor’ed together to make one missile enable register. Each bit in this register related to one of eight missiles that could be displayed on the screen at once. When the user fires a missile, the assembly code flips the corresponding bit in the user’s register which will cause the corresponding bit in the ‘xor’ed register to go high and when the missile reaches the top of the screen the hardware flips the corresponding bit in its register, causing the bit in the ‘xor’ed register to flip back to low. I have attached a few snippets of the required code to demonstrate this. The last major challenge of this project was finding a way to work well with my partner. We had a lot of difficulty coordinating our work for some of the other labs and I did find myself ultimately working on many of them alone. I took it as a challenge to find a way to divide the work appropriately. A way that allowed us to both show our individual strengths and to each work at our own pace, while still being able to come

together at the end and have a finished product that really displayed both of our efforts well. I feel that our design (and how we decided to split up the work for the final project) ultimately worked out well and was a great learning experience. It was a great learning experience in both RVfpga design as well as how to work well with others on such a large team project. Finally, I am including the link to my github repository for this project below:

https://github.com/apmaso/540ProjectFRe

```
54
55
56 always_comb begin
57 // Combinational logic for missles
58 // Every 1 M clocks (31.5 MHz clk -> ~0.03 sec) move missle up 2 pixels
59
60     // MISSLE 1 LOGIC
61     if (missle_en_xor[0]) begin
62         if (missle1_row_reg < 2)
63             begin
64                 missle1_row_next = player_row;
65                 missle1_column_next = player_column;;
66                 missle_en_next = (missle_en ^ (8'b00000001));
67             end
68         else
69             begin
70                 missle1_row_next = (missle1_row_reg - 2);
71                 missle1_column_next = missle1_column_reg;
72                 missle_en_next = missle_en;
73             end
74     end
75     else begin
76         missle1_row_next = player_row;
77         missle1_column_next = player_column;
78         missle_en_next = missle_en;
79     end
80
```

```
48
49      # If MSB is high --> Jump to toggling bits low
50      # MSB is bit 2 for now.... ('and' with 0x04)
51      # Need to update when adding the remaining missles
52      andi t2, a1, 0x80
53      bne t2, zero, not_missle8
54
55      andi t2, a1, 0x01
56      bne t2, zero, not_missle1
57      addi a1, a1, 0x01
58      sw a1, 0(t1)
59      beq zero, zero, bottom
60
61  not_missle1:
62      lw a1, 0(t1)
63      andi t2, a1, 0x02
64      bne t2, zero, not_missle2
65      addi a1, a1, 0x02
66      sw a1, 0(t1)
67      beq zero, zero, bottom
68
69  not_missle2:
70      lw a1, 0(t1)
71      andi t2, a1, 0x04
72      bne t2, zero, not_missle3
73      addi a1, a1, 0x04
74      sw a1, 0(t1)
75      beq zero, zero, bottom
```