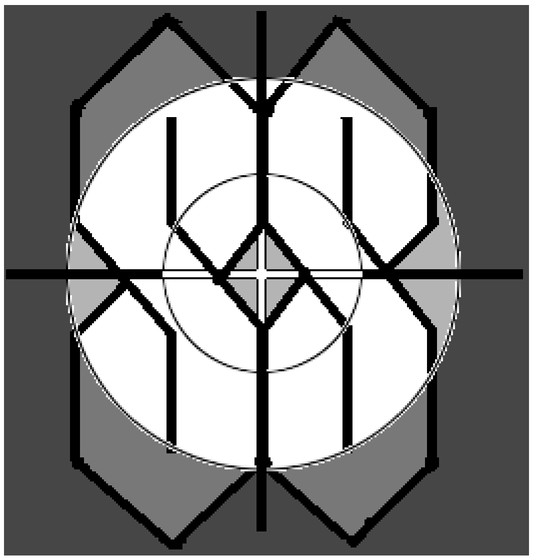
SHARP SHOOTER



ROHAAN AHMED **SVETISLAV PLEMIC**

Course:	ECE241 & ECE298
Professor:	Prof. Robert Irish, Prof. Hamid Timorabadi, Prof. Jonathan Rose
TA:	Dr. Michael Lapointe
Date:	December 5, 2007

1.0 Overview

The goal of this project was to combine the design skills gained in ECE 298 with the hardware programming skills acquired in ECE241 and demonstrate them by creating a design implemented in hardware. The project required that we use the techniques of Digital Systems Design to create a controlled-hardware system using a modular design methodology. That is, by using the Verilog Hardware Description Language (HDL) to program transistors and wires to form logic for outputs for hardware-control. With the availability of a wide variety of modern technology at our disposal, the design team decided to create a computer game that was fun and exciting to play with many unique features. After much debate and brainstorming [Appendix A – Brainstorming Mind Map], we decided to create 'Sharp Shooter', a game based on the same principles as the popular 1980's video game 'Duck-Hunt'. Essentially, this was achieved through a Field Programmable Gate Array (FPGA) microchip, which we programmed using a Computer Aided Design (CAD) application.

To achieve a high level of uniqueness, the team decided to include the following features.

- User interface using a reverse engineered Nintendo Zapper Light Gun.
- A target recognition system.
- A score keeping module.
- A unit to display the users' misses on the screen.
- A multi-player competition mode.
- Multiple levels that increase in difficulty as the score increases.

We will take a step-by-step approach to evaluating our design, starting with a breakdown of the components and operation and moving on to reverse engineering details, the testing and debugging of the design and finally the circuit optimization techniques we applied.

2.0 The Components Breakdown [Appendix B – Component Flow Diagram]

In order to create a versatile design, we first familiarized ourselves with a host of different hardware and software tools, extracting the maximum potential of the technology available.

2.1 Components list

- A Nintendo Entertainment System Light Gun (NES Zapper). [1]
- An Altera Quartus II Computer Aided Design (CAD) application.[2]
- An Altera DE2 Development & Education Board. [3]
- An Altera Cyclone II EP2C35F672C6 Field Programmable Gate Array (FPGA). [2]
- A Video Graphics Array (VGA). [4]
- A Red-Green-Blue (RGB) Display Cathode Ray Tube (CRT) Monitor.[5]
- 4 HEX-Displays (7-Segment Digital Displays). [3]
- A 390 KΩ (Kilo-Ohms) resistor.
- Random Access Memory (RAM).
- A Bitmap (BMP) to Memory Initialization File (MIF) converting application.
- A ribbon-cable pin expansion slot (JP1 or JP2).
- A Protoboard with a 5V and a Ground connection.
- A 50 MHz clock (alternates logic '1' & '0' 50 million times per second). [2]
- 17 Red Light Emitting Diodes (LEDs), 17 Switches and 4 Keys. [3]

One of the secondary objectives of our design was to make use of as many available components as possible. This included utilizing the various tools available on the DE2 Board and the Zapper, which was reverse engineered to make it compatible with a CRT monitor

3.0 The Operational Breakdown [Appendix C –System Layout]

To understand Sharp Shooter we must study the functionality of its many separate components.

3.1 The NES Zapper [Appendix D – The NES Zapper]

The function of the Zapper is to provide a triggering and detection mechanism. When a user pulls the trigger, the Zapper sends a '0' (1-bit) trigger signal to the FPGA for 50 milliseconds. The sensor signal, on the other hand, sends a '1' for 5 milliseconds when the mounted infrared sensor detects a 'drastic' change in the infrared light (drastic here implies black-to-white, red-to-blue and so on).

3.2 The State Machines

Sharp Shooter is a control intensive game. This means that there are various events that must be controlled either in series or in parallel. We achieved this through the implementation of Complex State Machines (CSM), which are FSMs with various sub-FSMs and multiple cases.

3.2.1 The Gun Interface CSM [Appendix E – The Gun Interface State Diagram]

This FSM was designed to detect a 'hit' using the Zapper's signals. Upon receiving a '0' trigger signal, the FSM sends a constant '1' signal to the Graphics Control CSM and waits for 50 milliseconds for a '1' sensor signal. If the time expires, the FSM resets. If, however, the FSM receives a positive sensor signal, it activates the Score Display and resets the Graphics Control.

3.2.2 The Graphics Control CSM [Appendix F – The Graphics Control State Diagrams]

The Graphics Control CSM displays and moves the target, and resets on a 'hit' or 'miss'.
There are two main states in which the Graphics Control FSM operates.

1. **Steady-State Operation**: The FSM first draws the target onto the screen pixel by pixel from the MIF file stored in the RAM. It then 'delays' for a duration depending on the current level of play. Next, the FSM erases the target and changes its location depending on a value from the Random Number Generator (RNG).
2. **Detect-State Operation:** In the detect state, the target's location remains constant and a white box is drawn over it for 50 milliseconds (to allow the Zapper to do its job).

3.2.3 The Score & Health Display FSMs [Appendix G & H – The Score & Health State Diagrams]

Both the Score and Health FSMs receive their inputs from a counter which is activated by the Gun and the Graphics Control CSMs respectively. The FSMs compare the counters' values with a look-up table, and perform the corresponding function. The Score FSM displays the score on the HEX-Displays whereas the Health FSM erases the health symbols from the health-bar.

3.2.5 The Game Control CSM [Appendix I – The Game Control State Diagram]

The Game Control CSM is the top-level FSM of the design. Its outputs directly activate the Graphics Control CSM, indirectly activating all other modules in the design.

3.2.6 The Random Number Generator

The random number generator operates as a counter operating on both edges of the clock. The RNG is always on, which means it constantly cycles through all the possible numbers. When a module accesses the RNG, it just sends back the most current value without stopping.

3.2.7 The Delayer Unit
The value of the delayer unit changes depending on the level of the game. When the score reaches a multiple of 10, the delayer unit decreases the delay duration, thus increasing the speed of the on-screen target.

4.0 Reverse Engineering the NES Zapper [Appendix J – Reverse Engineering the Zapper]

The Zapper was a pivotal piece of technology required in the game since it was the communicating medium between the user and the gaming system. The Zapper is factory designed to work with regular CRT TV monitors which have a much slower refresh rate (15 KHz) than CRT computer monitors (31 KHz) [6]. To test the Zapper's compatibility with our monitor, we created a program to rapidly switch the on-screen color between black and white. After running the test three times, we realized that, while the trigger signal behaved correctly, the sensor signal was faulty, that is, it did not send a '1' signal at every color change. After much research [6] and consultation with the teaching assistants it was found that the speed at which the sensor was operating was too slow when compared to the CRT monitor's refresh rate. At that point we decided to insert a 390 KΩ resistor in parallel with an existing 390 KΩ resistor (giving a 195 KΩ equivalent resistor) to speed up the signal. We ran the same test again and found that the modification had worked, in that, it sped up the sensor just enough to allow it to detect all color changes.

5.0 Project Management and Scheduling [Appendix K – Project Management]

A schedule was created at the beginning of the design process for the timely completion of the project. We created a Gantt chart created after the completion of the project to map out our success.

6.0 Testing and Debugging [Appendix L – Test & Simulations]

Due to the modular nature of the game, Sharp Shooter was designed using an incremental design methodology. This enabled us to test each module separately for its functionality. Since the beginning of the semester, we had found that functional and timing simulations (using Quartus II) were an extremely useful way to test designs, and therefore, we decided to make simulations our primary testing technique. Apart from simulations, we also created test-programs when required to test the functionality of each hardware component separately.

6.1 Testing

Each module and FSM in the design was first tested individually and then tested in combination with other modules.

6.1.1 The Zapper: To test the Zapper, we connected each of the sensor and trigger signals to 2 different LEDs on the DE2 Board. We then wrote a test program to see if the lights lit up when the trigger was pulled and the screen color was changed. Next, we ran simulations of the gun to test the behavior of various combinations of inputs and study their outputs.

6.1.2 The FSMs: All the FSMs in the program were first simulated individually by assigning their inputs to switches and output to the LEDs. Once the desirable outputs were achieved, we proceeded to combine the CSMs and FSMs and simulate their combined functionality.

6.2.3 The VGA: The VGA was tested in 3 phases by writing 3 separate test programs. The first program displayed a static picture onto the screen using a MIF file. The second moved a picture around the screen using the DE2 keypad. The final program moved the target around the screen constantly; 'bouncing' it off the edges by reversing its direction each time the end of the screen was reached. These programs allowed us to fully understand the functionality of the VGA.

6.2 Debugging

The purpose of testing was to detect any glitches in each individual module. One very successful way our team solved errors was by using the method of 'code review', where the partner of the person designing the module would look at the circuit diagram and the Verilog code and try to understand what is being done. 90% of all our errors were resolved by this method. Problems that were not solvable by code review took significantly longer to resolve. One major problem we faced during the design stage was that when we tried to display a white box in place of the target in the 'detect' stage of the Graphics Control CSM, we were only able to print one pixel while the rest of the picture remained unchanged. After a day's work, we realized the problem was because the offset was not being incremented. Once this information was known, we created a special offset-adder module and connected it to the CSM, which solved the problem. Another problem we faced during design was that our initial random number generator function was too time-consuming due to the various division and multiplication functions we were using to generate the random value. The module implementation was so slow it was taking longer than the clock period allowed, that is, our maximum allowed frequency (fmax) went down to 14MHz (the fmax must be greater than 50MHz for the game to work correctly). This forced us to re-think our process and design a much simpler, and equally capable, RNG by utilizing both edges of the clock ('1' and '0') and by keeping the RNG constantly running (fmax = 134.7MHz).

7.0 Circuit Optimization [Appendix M – Circuit Optimization]

One of the primary tasks of an engineer is to make things simpler and more efficient. Therefore, despite there being no official restrictions on the cost of the design, we tried very hard to decrease the cost and increase the fmax of the design using the following techniques.

7.1 Karnaugh Maps: This technique allowed us to reduce the number of gates used [2] , and therefore increase the speed, of the Score and Health Display modules. For example, the function that tells segment-5 of the HEX-Displays to turn on initially included 5 gates (4 OR gates and 1 AND gate). After implementing the Karnaugh Map for the same function, we reduced the cost to 4 gates (3 OR gates and 1 AND gate). This is a significant gain because the transformation reduced the propagation-gate-delay by 1 OR gate, which is approximately 2ns, giving us a surplus of 1/10th of a 50MHz clock period (resulting in a 5MHz increase in fmax).

7.2 Re-structuring: Utilizing this technique enabled us to simplify the implementation path of our circuitry, which reduced compile time. It also assisted with 'code review' debugging due to the easy-to-follow re-structuring of all modules. It was most helpful when trying to reduce the number of ‘inferred latches’ (multiple assignments of a variable in the same block) in our program, which have the potential to reduce the fmax by half.

7.3 Parallel Calculations: This technique takes advantage of the benefits of hardware over software. In hardware, it is possible to transmit information simultaneously through separate wires, reducing the overall run-time of the circuit. For example, we changed the offset values of the target in the vertical and horizontal directions simultaneously creating a fluid animation.

7.4 Resource Sharing: In our program we have used the same inputs and outputs to activate multiple modules, sharing the resources. Implementation of this technique also assisted with parallel calculations.

7.5 Quartus ‘Try Harder’: Lastly, we commanded Quartus to ‘try harder’ to compile a more efficient circuit [7].

8.0 Success of the Design

The design was successful in that it achieved its main objectives. We managed to take previously engineered goods and combine them with custom created circuitry to develop Sharp Shooter. We decided to take a spiral/incremental model design approach. For example, we first made the background appear onto the screen, then created a target and then made the target move. On top of this platform we constructed the rest of the design. Our parallel design strategy allowed us to spend a significant amount of time testing and debugging the project. Good team chemistry allowed us to diagnose problems and debug them quickly using the ‘code review’ method of debugging. Keeping this in mind, it must be noted that the design was not a complete success. While connecting the Zapper to the DE2 Board, we failed to follow the basic rule of reading the manual. As a result, on the day of the design presentation, we connected the Zapper to the high-voltage source for too long and overheated the circuit inside, breaking the circuitry and rendering it unusable [Appendix N – Game Screenshots].

9.0 Conclusion

Sharp Shooter is a success in that it accomplished everything we set out to do. The project, in large, was a beneficial exercise because it allowed us to become creative, by allowing us to use the hardware technology available to develop our own game, which is something we had never been introduced to before. We utilized all the engineering skills acquired throughout the semester in ECE298 and ECE241, such as creative design techniques, communication skills, team-work/interpersonal abilities, reverse engineering skills and so on, to create a digitally controlled gaming system using hardware based technology.