Jason Glass, Austin Ryder and Nihkil Vishnubhatla

EECS 355 Tank Project Winter 2017

# ­Design Process & Methodology

The goal of this design project was to demonstrate understanding of VHDL and FPGA systems through the design and implementation of a tank battle videogame. The game must demonstrate proper usage and understanding of I/O, asynchronous vs synchronous operation, modular design and correct coding standards.

Our team started by building out the necessary bottom-level modules we needed. Then we worked from the top-down to build our Tank game process hierarchy. We tested and synthesized the game in incremental steps along with version control through .Git to ensure that our workflow and progression was stable and clean.

We first focused on core functionality like tank movement and collision detection before implementing peripheral improvements like a score tracker and victory screen. Our goal was to use modularity throughout the project and make scaling up as easy as possible by exploiting loops and generics.

The requirements for the tank game were as follows:

* Draw two tanks of different colors that face each other on the VGA screen
* Tanks should continuously move in only the x-axis with 3 different user-controlled speed settings.
* Tanks can fire one bullet at a time that scores them 1 point if it hits the enemy tank.
* The LED screen should track the player scores.
* When a player scores 3 points a victory message should be set on the LCD screen while only the winners tank displays.
* There should be a global reset to the game.

# System Architecture and Components/Processes

Team Files

* **Bullet\_hit**
  + Outputs whether a bullet has scored a hit by comparing the bounding box of the bullet and the enemy tank through generation of the collision detector module
* **Bullet\_position**
  + Process to track the y-coordinate of the bullet and also ensure that only one bullet can be fired at a time
* **Collision\_detector**
  + Takes in two bounding box objects and determines if they have a collision
* **integer\_register**
  + a register for integer values
* **tank\_clock**
  + the clock divider process that uses an accumulator to slow down the drawing of the tank\_game to a speed that ensures optimal game feedback and responsiveness relative to human speeds.
* **tank\_game**
  + top level file that controls the game and has the instantiation of necessary modules
* **top\_tank**
  + uses a speed variable to update the x\_position of a tank relative to an input position
* **VGA\_top\_level**
  + Contains top\_level logic for VGA processes
* **Coordinate\_register­**
  + a register for our user-declared coordinate type, a tuple for (x,y)

Given Files:

* **colorROM**
  + read-only memory for Colors
* **de2lcd**
  + decimal to lcd screen
* **Keyboard**
  + Take in keyboard inputs
* **Leddcd**
  + Pass text to the board LED screen and draw on 7-segment display
* **Oneshot**
  + Ensures only one bullet fires at a time
* **pixelGenerator**
  + Logic that draws all objects on the screen
* **ps2**
  + handle input from keyboard
* **vga**\_**sync**
  + control the refresh frequency of the VGA to the display

Design Choices:

We chose to implement a number of design choices or optimizations into our design. They are as following

1. Clock Divider
   1. Because the base clock would refresh the screen too fast for a human player to play properly we utilized a clock divider that sends a “Draw Ready” signal to our system on a multiple of the clock to effectively slow it down while still maintaining computational throughput.
2. Augmented Keyboard I/O
   1. We noticed that the given hist\_signal implementation of the keyboard I/O module introuduced an element of Asynchronous blocking into our project. We decided to eliminate the scan for F0\_KEY on Key\_UP and instead just scanned for the KEY\_DOWN code while implementing a key\_blocker that prevents a user from just holding a key to send multiple controls. A key will only activate once per press now with no blocking.

# Board Implementation and Peripherals

* Altera DE2-115 Board
  + A development and educational board equipped with 114,480 Logic elements (LE), up to 3.9 Mbits of RAM and 266 multipliers.
  + Utilized the VGA I/O, LCD screen, LED 7-segment and DIP switches for this project

# Synthesis Results, including memory, clocks and resource utilization

Analysis and Synthesis Results

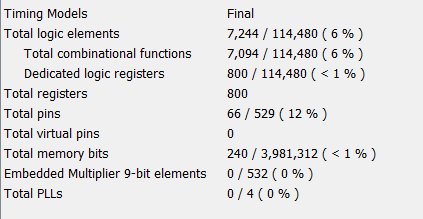


Figure 1: Registers, LE’s and Memory utilization

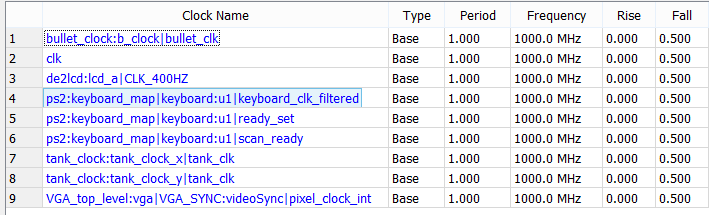


Figure 2: Timing Analysis – Clock List

TOP LEVEL RTL

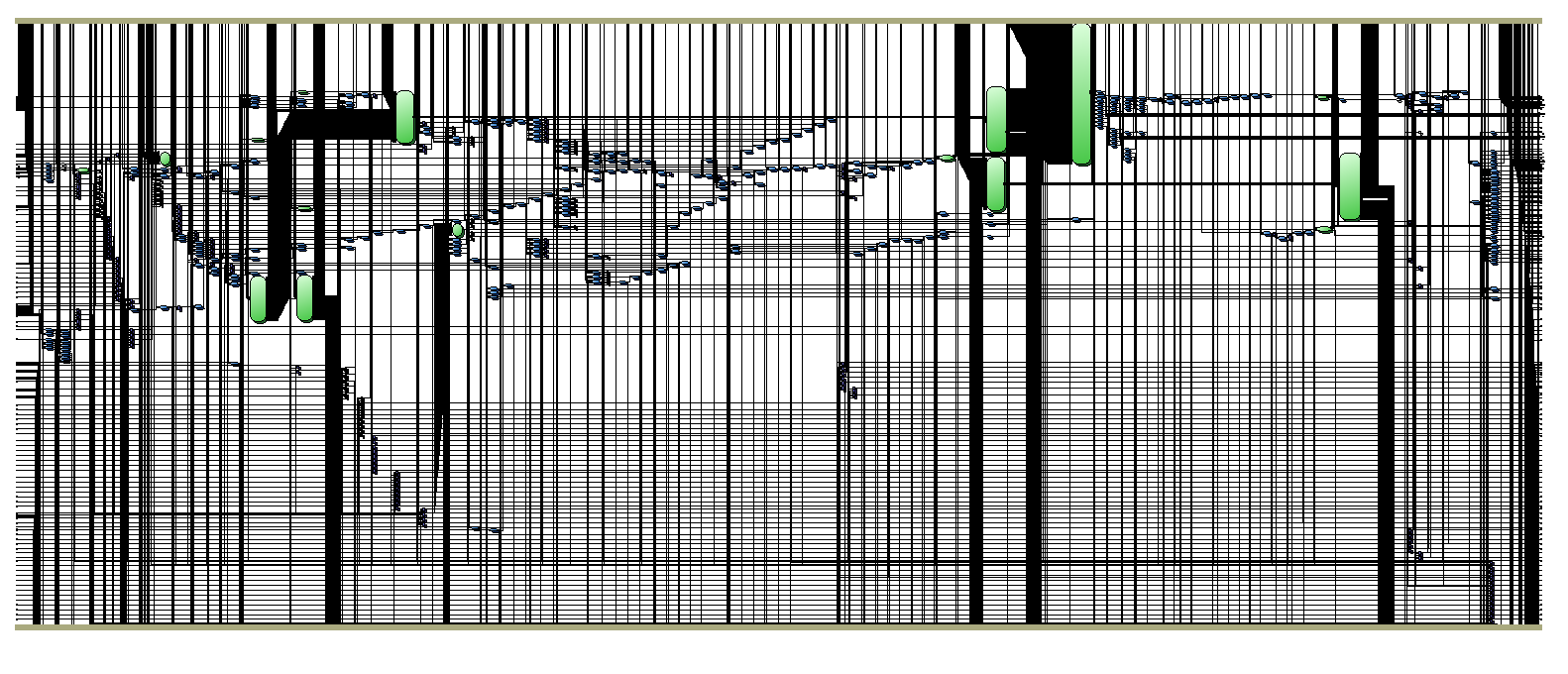


Figure 3: Top-Level RTL

Conclusion

Our final implementation of the game worked and had all required functionalities and gameplay elements. We encountered a variety of difficulties along the way such as inferred latching, timing conditions and asynchronous vs. synchronous dependency resolution. Future optimizations we would like to apply include expanding the tank gameplay to allow different bullet types and also tank rotation. We would like to clean up our synchronous logic to try to reduce the number of logical elements we require for the game calculations.   
 Our architecture became more complicated as the project developed over time and it would be ideal to start a v.2. of this project and rebuild some of the VGA functionality along with Pixel Generator to produce a more streamlined system. Many of our latching issues came about from long chains of conditional logic that could be implemented with safer and faster Case statements.   
 We are definitely happy with our final product however and are excited to have learned important soft and hard-skills for the technology world and FPGA devices alike. Although we heavily deviated from our design goals and desire to write clean-optimized code, the reality of development struck and we were forced to use hack-y rough solutions. While the results were still valid we believe it is important to adhere to coding standards whenever as possible, for both code quality and improvement to one’s own coding repertoire.

# Appendix A: Package Entity

-- Tank Game Package

library IEEE;

use IEEE.std\_logic\_1164.all;

package tank\_package is

constant BULLET\_SPEED : integer := 2;

constant BOUNDING\_BOX\_CORNERS : natural := 4;

type bounding\_box is array(0 to BOUNDING\_BOX\_CORNERS - 1) of integer; -- (x\_left, x\_right, y\_top, y\_bottom)

type coordinate is array(0 to 1) of integer; -- (x, y)

type size is array(0 to 1) of integer; -- (width, height)

constant TANK\_SIZE : size := (120, 100);

constant BULLET\_SIZE : size := (10, 10);

component top\_tank is

port(

clk : in std\_logic;

rst : in std\_logic;

score : in integer;

x\_out : out integer);

end component;

component ps2 is

port( keyboard\_clk, keyboard\_data, clock\_50MHz ,

reset : in std\_logic;

scan\_code : out std\_logic\_vector( 7 downto 0 );

scan\_readyo : out std\_logic;

hist3 : out std\_logic\_vector(7 downto 0);

hist2 : out std\_logic\_vector(7 downto 0);

hist1 : out std\_logic\_vector(7 downto 0);

hist0 : out std\_logic\_vector(7 downto 0)

);

end component;

component tank\_clock is

port (

clk : in std\_logic;

reset : in std\_logic;

speed : in integer;

tank\_clk : out std\_logic

);

end component;

component bullet\_clock is

port (

clk : in std\_logic;

reset : in std\_logic;

bullet\_clk : out std\_logic

);

end component;

component bullet\_position is

port (

clk: in std\_logic;

reset: in std\_logic;

direction: in std\_logic;

bullet\_fired: in std\_logic;

bullet\_hit: in std\_logic;

current\_bullet\_exists: in std\_logic;

current\_bullet\_position : in coordinate;

current\_tank\_position: in coordinate;

new\_bullet\_position: out coordinate;

new\_bullet\_exists: out std\_logic

);

end component;

component collision\_detector is

port (

bounding\_box\_a : in bounding\_box;

bounding\_box\_b : in bounding\_box;

collision : out std\_logic

);

end component;

component bullet\_hit is

port (

bullet\_position : in coordinate;

enemy\_position: in coordinate;

hit: out std\_logic

);

end component;

component integer\_register is

port(

clk : in std\_logic;

int\_in : in integer;

int\_out : inout integer);

end component;

component coordinate\_register is

port(

clk : in std\_logic;

int\_in : in coordinate;

int\_out : inout coordinate);

end component;

component std\_logic\_register is

port(

clk : in std\_logic;

int\_in : in std\_logic;

int\_out : inout std\_logic);

end component;

end package tank\_package;

package body tank\_package is

end package body tank\_package;