**Tank:**

As described in the project description, the tanks move horizontally along the screen at three different speeds controlled by the keyboard. When the edge of the tank reaches an edge of the screen, the tanks bounce back. The inputs for the tank entity are: clk, reset, pulse, speed, isLoser. The one output is the new x position, xout.

entity tank is

generic (

width : std\_logic\_vector(5 downto 0)

);

port (

clk : in std\_logic;

reset : in std\_logic;

pulse : in std\_logic;

speed : in std\_logic\_vector(1 downto 0);

isLoser : in std\_logic;

xout : out std\_logic\_vector(9 downto 0)

);

end entity;

The tank movement was implemented with a finite state machine that consisted of four states: idle, update, lost, and waitOnPulseLow. The idle state, as its name implies, does nothing until a pulse is detected, in which case it moves to the update state. The tank waits for a pulse rather than a clock because the VGA monitor has a much slower frequency than the FPGA’s clock, so we want the movement to be updated in the VGA monitor’s time domain, so that the tank moves at the proper speed. In the update state, the tank’s position is updated based on the speed. The tank has an internal move direction that is 0 if the tank is moving in the positive x direction, and 1 if the tank is moving in the negative x direction. Based on this move direction, the tank either adds or subtracts the speed times some speed factor set in tank\_pack.vhd to its position and outputs this as xout. The width generic, is used to check if the tank is going to hit one of the edges. In the case that the tank does hit an edge, the move direction is inverted, and the tank then moves in the opposite direction to achieve the bouncing effect. From the update state the, tanks either goes to the waitOnPulseLow or the lost state. The tank will advance to the lost state if the isLoser input is 1. In this state the tank position is set so that only the winning tank is drawn on the VGA monitor. Otherwise, if isLoser is 0, the tank moves from the update state to the waitOnPulseLow state. This state is necessary because we want the tank to only update once per pulse, and since the pulse remains high for longer than one clock cycle, sending the tank to back to the idle state before pulse is low would cause the tank position to update multiple times. In the waitOnPulseLow state, the next state is always waitOnPulseLow, unless the pulse input is 0, in which case next state is idle.

**LCD:**

The LCD display on the FPGA board was used to display the winning player. It follows the de2lcd mini project closely with a few modifications to meet our needs. The entity is declared as shown below:

ENTITY de2lcd IS

PORT(reset, clk\_50Mhz : IN STD\_LOGIC;

winner : IN STD\_LOGIC\_VECTOR(1 downto 0);

LCD\_RS, LCD\_E, LCD\_ON, RESET\_LED, SEC\_LED : OUT STD\_LOGIC;

LCD\_RW : BUFFER STD\_LOGIC;

DATA\_BUS : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0));

END de2lcd;

The de2lcd consists of a state machine with clear, reset, and write character states. For our purposes, we wanted the LCD to display either “Player 1 Wins!” or “Player 2 Wins!” or nothing depending on the state of the game. Because of this, a two bit input called “winner” was added to the original de2lcd component which controls some of the state transitions to print the correct message. Player 1 wins, player 2 wins, and nobody has won yet correspond to the bit patterns 01, 10, and 00 respectively. The two messages that need to be displayed differ by only one character, so there are only 13 write character states needed to print the messages. At the state that writes the character that differs (the number of the player who wins) the appropriate character is selected based on the winner input. In order to print nothing on the screen, the MODE\_SET state has an if statement that either advances to the first write character state or the RETURN\_HOME state depending on the winner input. If the winner input is 00, the next state is RETURN\_HOME which skips all the write characters.