**Bullet:**

The entity for the bullet component is described below:

|  |
| --- |
| entity bullet is |
|  | generic ( |
|  | tank\_y\_pos : std\_logic\_vector (9 downto 0); |
|  | bullet\_direction : std\_logic |
|  | ); |
|  | port ( |
|  | clk : in std\_logic; |
|  | reset : in std\_logic; |
|  | pulse : in std\_logic; |
|  | fire : in std\_logic; |
|  | tank\_x\_pos : in std\_logic\_vector (9 downto 0); |
|  | collision : in std\_logic; |
|  | next\_bullet\_active : out std\_logic; |
|  | next\_y\_pos : out std\_logic\_vector (9 downto 0); |
|  | next\_x\_pos : out std\_logic\_vector (9 downto 0) |
|  | ); |
|  | end entity bullet; |

Like with every component, the bullet takes a clock, reset, and a pulse input. It also takes an input signal called “fire” which is a signal coming from the keyboard indicating when a bullet should be shot, the x-position of the tank, indicating where the bullet should originate when shot, and a collision input indicating if the bullet has collided with the opposing tank and should be eliminated from the screen. The bullet component outputs whether it is active or not and its next x and y position after each pulse. The basic structure of the component is a simple state machine. In the clocked process, the current x, y, and active signals are updated with their corresponding next signals. The combinational process is made up of two states: the idle state and waitOnPulseLow state. The idle state handles all of the logic. In this state it is first checked whether a bullet has been fired, if so a bullet is fired from the tank’s current x position. If a bullet is active the bullet position is updated. Finally, if the bullet has not been fired and it is not active it stays off screen. After these initial checks, the bullet direction is verified in order to update the direction properly and then the collision bit is checked to see if the bullet should remain active or disappear. The waitOnPulseLow state simply ensures that the idle state doesn’t keep occurring on each clock cycle by sampling waiting until the overall pulse signal goes low.

**Score:**

The entity for the score component is described below:

|  |
| --- |
| entity score is |
|  | port ( |
|  | clk : in std\_logic; |
|  | reset : in std\_logic; |
|  | collision : in std\_logic; |
|  | pulse : in std\_logic; |
|  | score : out std\_logic\_vector(3 downto 0) |
|  | ); |
|  | end entity; |

The score component also takes a clock, reset, and pulse input. It also takes a collision bit input and outputs the current score. The basic structure for the score is a 2-state state machine. The clocked process simply updates the current score and current state. The two states in the combinational process are the idle state and waitforlow state. In the idle state, the score is updated by one if the collision bit is high while the pulse is high. If not it transitions to the waitforlow state and in this state the component just waits for the next pulse high sequence.