

**School of Electronic**

**and Electrical Engineering**

**ELEC5566M: FPGA Design for System on Chip**

MINI - PROJECT  
**Videogame:** READY, STEADY, BANG!

|  |  |  |
| --- | --- | --- |
|  | STUDENT 1 | STUDENT 2 |
| NAME | **SANJITH CHANDRAN** | **SHRAJAN BHANDARY** |
| ID | **201290793** | **201289634** |

**READY, STEADY, BANG!**

**1. ABSTRACT**

This report shows the use of Behavioural Verilog HDL and concepts learnt in the ELEC5566 course for the design and development of our mini-project, which was inspired by a video-game called “**Ready, Steady, Bang!** [1]”. This project was completed using Quartus to build various modules and test benches as well as utilizing up-to two DE1-SoC boards for final testing and verification through hardware checks.

**2. INTRODUCTION**

The videogame can be compared to a “cowboy-showdown” where there are two players i.e. gunmen, who are directly opposite of each other at a fixed distance and are unable to move. The goal of both players is to shoot the opposing player first. It should be noted that the person playing the game is not responsible for the accuracy of the shot taken i.e. 100% accuracy. The game starts with an initial screen which advertises what the game is called and how each character is depicted. Then, by using the touch feature of the LCD, one touch brings up the option for playing the game against either the computer or another human. This option is selected by using the keys/buttons: two buttons for moving the selection box to the left and to the right while the third button is for confirming the choice of game mode. Once the game mode is selected, each round is started by the following words appearing individually, sequentially and in the middle of the screen: “**READY**”, “**STEADY**” and “**BANG**”. The first two words are separated by a fixed delay whereas the word “**BANG**” shows up at a random delay of 2 to 10 seconds. As soon as the word “**BANG**” is displayed, by touching anywhere on the LCD display, each player must shoot their opponent. In the 1P mode, the computer takes up to 1 second to shoot. The first person to register their shot is the winner of that round and earns 1 point, with the losing character’s head being replaced by a red “X” to indicate a loss. The score can be seen on the seven- segment display and is updated after each round. The round then ends by the word “**NEXT**” showing up in the middle of the screen and the 1st player must touch their LCD screen to start the next round.



**Figure 1:** Main screen of the game.

The overall game winner is decided by the first person to win 5 points, whereby the winner is indicated in the middle of the LCD screen. i.e. either “**WINNER P1**” or “**WINNER P2**”. However, there is one caveat to winning each round. The shot should only be taken once the word “**BANG**” shows up in the middle of the screen. If the shot is taken after the word “**STEADY**” disappears and before the word “**BANG**” appears, then this is a point for the opposing player and the player who takes the premature shot has his character’s head replaced by a red X to indicate a loss that round. It is also possible for both players to be killed simultaneously but this is quite a rare scenario. When a winner has been determined, the game can be restarted using a slide-switch. For the 2P mode, a secondary DE1-SoC board is connected via the spare GPIO-1 pins and utilizes an additional LCD display for the 2nd player to register their inputs for shooting their opponent during the game.

**3. THE GAME REVEALED**

Ready steady bang is a multi-player game that was developed by employing a hierarchical design that instantiates multiple submodules. The low-level entities perform different, albeit crucial functions to ensure that the gameplay is smooth and executes without any glitches. The primitive submodules used are: N-bit counter, N-bit comparator, Up-counter and Seven-segment display unit. In addition, the game utilizes sophisticated IP cores such as: Random Number Generator, Frequency Divider, LT24 Display Module, Game Engine and Video Engine.

**3.1. Random Number Generator Submodule**

A salient feature of the game is the requirement for players to have rapid reflexes and immense patience to score points. The instant the word “**BANG**” appears on the screen, players must aim to tap their respective LCD screens before their opponent. Therefore, the appearance of “**BANG**” after “**STEADY**” has to surface after a random and unpredictable time delay. Linear Feedback Shift Registers (LFSRs) can be used to generate the required random numbers. However, LFSRs are pseudo-random in nature [2]. The output of an LFSR repeats after a certain period of time and therefore showcases a predictable behaviour. In order to overcome this drawback, the random generator utilizes a counter that increments indefinitely at every positive edge of a clock that runs at an extremely high frequency. The value of the counter register is selected as the bang delay and is determined by certain instances such as the tapping of the LCD screen.

**A screenshot of a computer

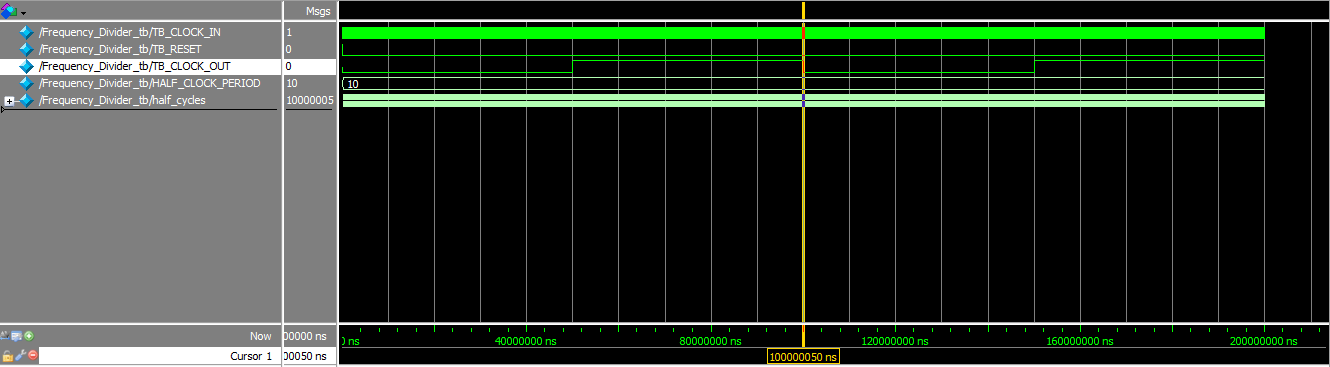
Description generated with very high confidenceFigure 2:** Input and Output Waveforms of submodule Random Number Generator.

The combination of a high frequency clock along with the unpredictable tapping of the screen generates a true random number. The random number generator has a minimum value that ensures that the “**BANG**” flag doesn’t occur too fast for the players to react to. Furthermore, there is a maximum value so that players do not have to wait anxiously for long periods of time.

**3.2. Frequency Divider Submodule**

The Cyclone V FPGA board has a 50MHz clock that has a time period of 200ns. Making use of this clock to count the bang delay requires a large amount of computation power. A frequency divider circuit is implemented in order to simplify this process. The frequency divider circuit takes the incoming clock signal of 50 MHz and divides it by the required frequency of 10Hz. The frequency divider circuit is implemented using an N-bit counter that counts to a maximum value called the Toggle-value. The Toggle-value is half the ratio of incoming clock frequency to the desired frequency. An output clock signal is generated to switch (LOW to HIGH or vice-versa) whenever the Toggle-value is reached. This is carried out by the N-bit comparator module. The equation for the toggle value is given by:

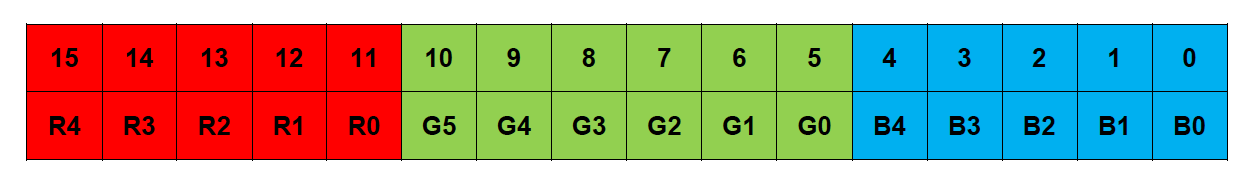
Toggle value = ……… ……………………… (1)



**Figure 3:** Input and Output Waveforms of submodule Frequency Divider with input clock of 50 MHz and output clock of 10Hz using ModelSim.

**3.3. Video Engine Submodule**

The graphics for the game were derived from the LT24Top.v module that was provided in Lab 5 of this course. It was noted that *pixelData* is a 16-bit register that controls the colour of each pixel based on the RGB scale. However, instead of creating a gradient of colours as shown in that lab, 16-bit hexadecimal colour codes were used.



**Figure 4:** 16-bit representation of the LCD pixel data.

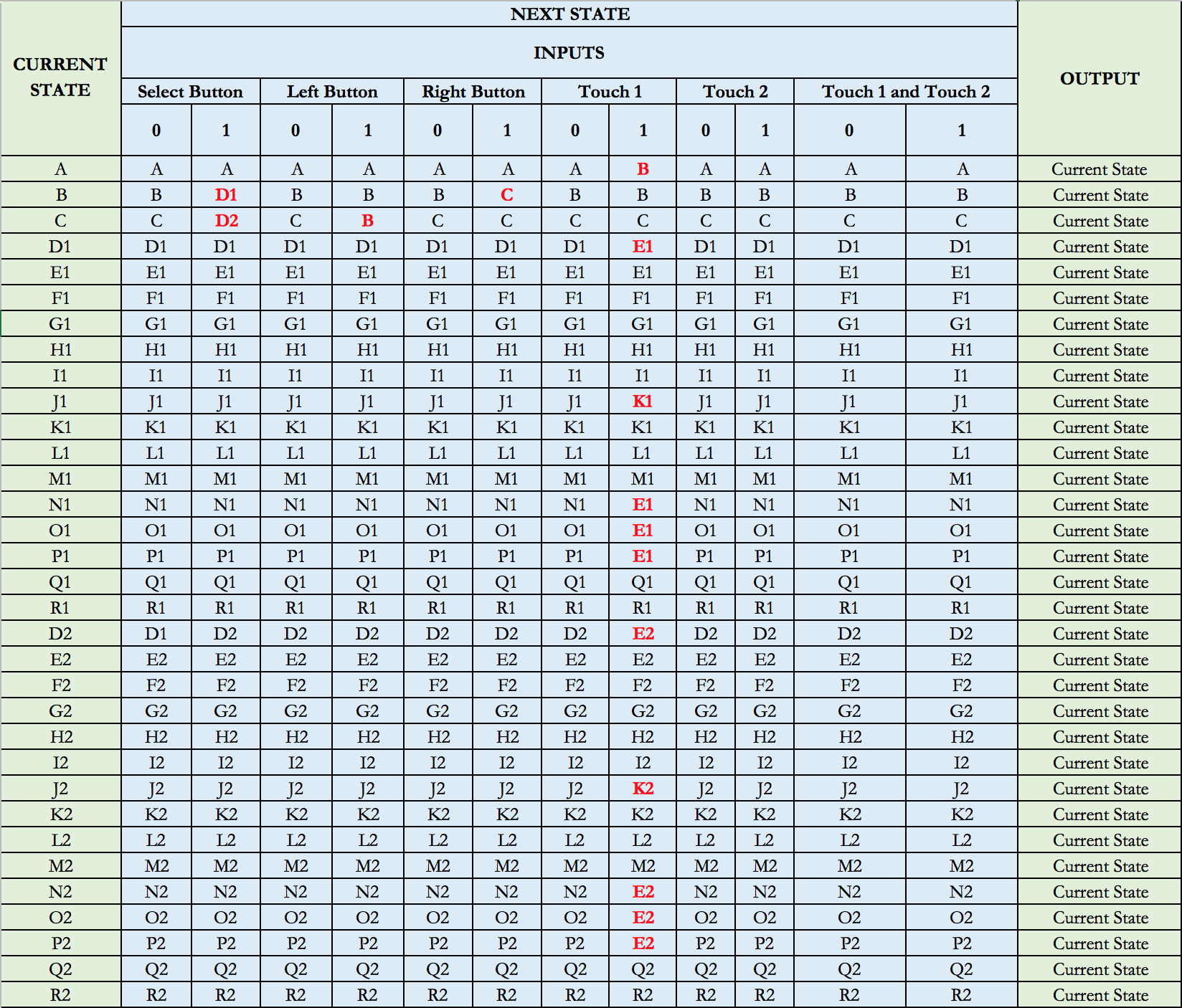
There are two inputs that determine the location of which pixels are coloured: lcd\_pixel\_y\_address and lcd\_pixel\_x\_address. These inputs, along with *if-statements* were utilizedto determine which pixels should be filled with a particular colour i.e. drawing the scarf, gun or pants of the gunmen characters. These checks are performed across the entire 240 x 320 LCD screen. At the end of these if statements, the unused pixels are set to a default colour of white or black so as to avoid errors when changing graphic displays during each stage of the game. The graphics were also parametrized by giving each element its own local axis i.e. letters of words and segments of the gunman character such as head and pants. This was done so that moving the location of the gunmen imagery or entire words could be done by changing two values: the x & y starting position of the local coordinate systems i.e. GUNMAN\_X & GUNMAN\_Y. These local parameters are set at the beginning of each state machine description. An example of this can be seen in Appendix B. It should be noted that the shapes of letters and structure of parts of the gunmen were derived using an excel sheet that comprised of 240x320 square cells. This proved an effective way of determining the logic of drawing shapes and lines.

|  |  |  |
| --- | --- | --- |
| **State Name** | | **Name of the screen** |
| A | | Main screen of the game |
| B | | Single Player selection screen |
| C | | Multiple Player selection screen |
| D1 | D2 | Start screen |
| E1 | E2 | Empty Screen 1 |
| F1 | F2 | Ready Screen |
| G1 | G2 | Empty Screen 2 |
| H1 | H2 | Steady Screen |
| I1 | I2 | Empty Screen 3 |
| J1 | J2 | Bang Screen |
| K1 | K2 | First player kill screen |
| L1 | L2 | Second player kill screen |
| M1 | M2 | Both players kill screen |
| N1 | N2 | Next on First player kill screen |
| O1 | O2 | Next on Second player kill screen |
| P1 | P2 | Next on Both players kill screen |
| Q1 | Q2 | First Player Winner Screen |
| R1 | R2 | Second Player Winner Screen |

**Table 1:** Different screens of the game that correspond to a particular state. States from D to R have suffix either 1 or 2 that denote single player mode or two player mode.

**3.4. Game Engine Submodule**

This submodule acts as the main control centre of the game. It obtains different user inputs via hardware peripherals and decides each stage of the game. Every single screen in the game is considered as an individual state. The Game Engine Submodule instantiates multiple low-level modules such as the N-bit comparator, N-bit counter, Random Number generator and Frequency Divider circuit. The Game Engine also implements a Moore model to structure the overall gameplay of “**Ready, Steady, Bang**”. The game consists of 35 states and every state is determined by the changes in the inputs, players’ scores and the current state. Each state represents a screen in the game and has a unique number associated with it. These states are delineated in the tables below:



A screenshot of a cell phone

Description generated with very high confidence

A close up of a building

Description generated with high confidence

**Table 2:** State tables of the game.

**3.5. Ready Steady Bang**

The top-level entity of the game is parameterized and designed hierarchically to implement different versions of the game with minimal changes. This module is the main control panel of the game. It instantiates multiple submodules such as:

* LT24 display module – to generate required images/patterns on the screen.
* Video Engine – to generate the pixel data that determines the shapes and sizes of the patterns based on the gameplay.
* Game Engine – Controls and coordinates all user inputs, processes them to determine game states and keeps track of player scores.
* Seven Segment Controller – Displays the score of each player playing the game.

All the control inputs and screen states are manipulated to create game-play. The inputs are fed to the game engine that determines the state and passes this information to the video engine. Along with state values, it also passes player scores to the seven segment display controllers. The video engine accepts the state values from the game engine and generates corresponding pixel data depending on the co-ordinates. This pixel data is fed to the LT24 display module that then showcases different screens and makes up for the complete gameplay.

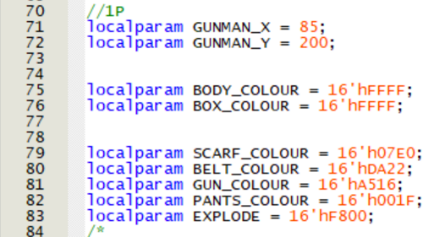
**4. SOFTWARE IMPLEMENTATION AND TESTING**

**4.1 Video-Engine Test Bench**

Test benches were created to test various sections of the project. An example of this can be seen below with respective sections of code and the resulting signals from the Model Sim program that indicate a change in the status of output signals based on their inputs. For example, the pixelData value is changed to green (Hex Value = x07e0) at a range of specific x & y address values [3]. As it can be seen in the code, if the x-value (xAddr) lies between GUNMAN\_X+4 and GUNMAN\_X+13 while the y-value (yAddr) is GUNMAN\_Y+15, the pixelData should be green. Based on the code,

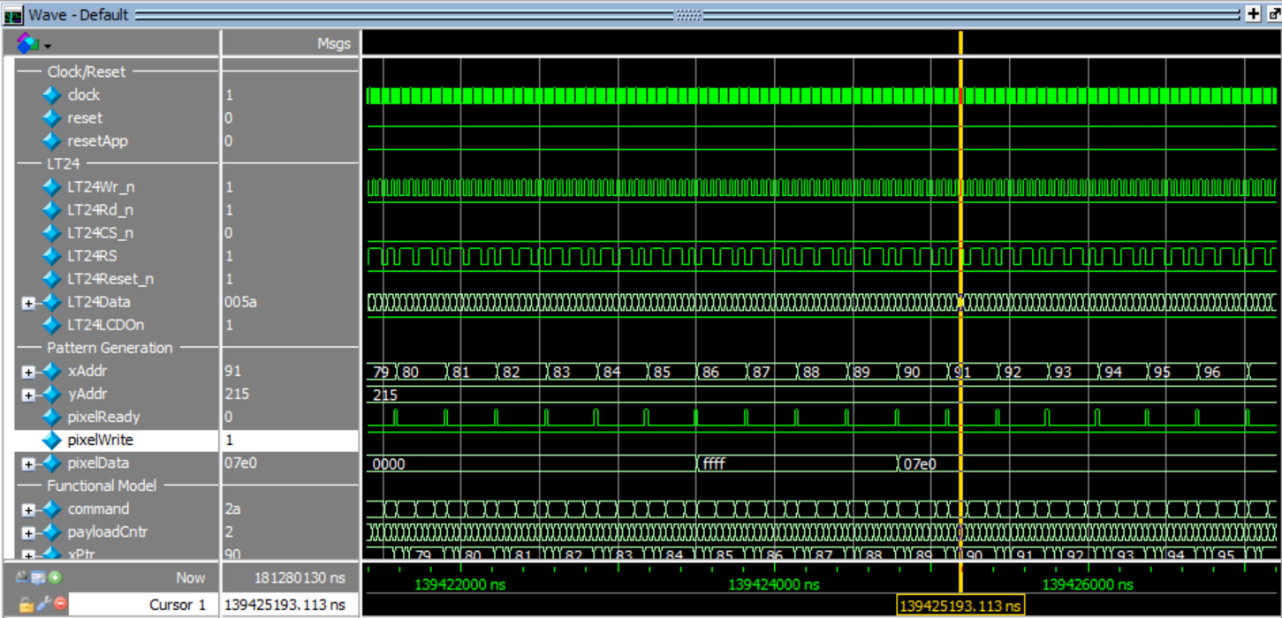
* GUNMAN\_X+4 = 85+4 = 89
* GUNMAN\_X+13 = 85+13 = 98
* GUNMAN\_Y+4 = 200+15 = 215

And as seen in the Waveform diagram, since the xAddr value of 91 and yAddr value of 215 meets the above conditions, the pixelData is set as green. Additional code and test results can be seen in the Appendix B for testing the location of letters and setting the colour of these letters to white i.e. pixelData value is set to xFFFF. Pixels that are not used to drawing game characters segments or letters, are set as black (x0000). It should be noted that these white words and black backgrounds are only used for the initial game screen while during the game the background is set as white while the words are set as black.



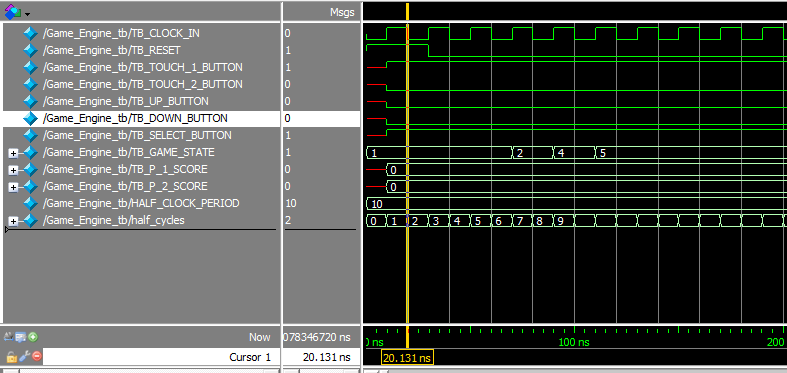


**Figure 5:** Code snippets that set values for individual pixel data of LCD based on its x-y coordinate.

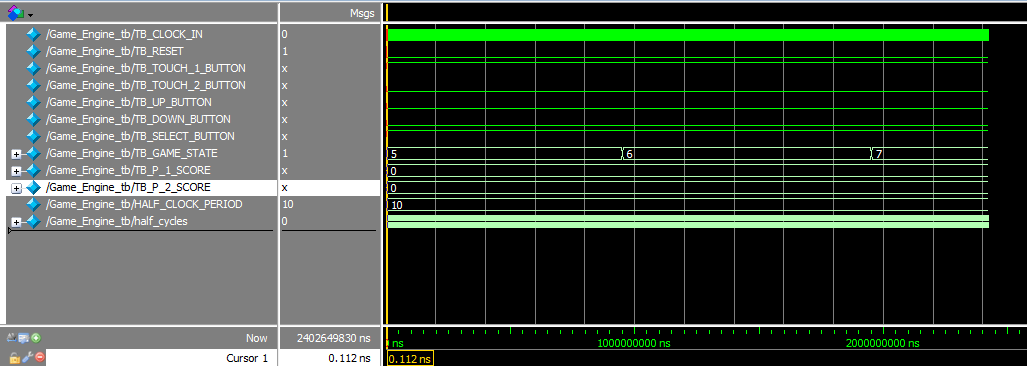


**Figure 6:** Input and Output Waveforms of video engine with different pixel data values.

**4.2 Game-Engine Test Bench**



**Figure 7:** Input and Output Waveforms of game engine with clock of 50 MHz, touch and button inputs along with player scores and game states (1-5) as outputs.



**Figure 8:** Input and Output Waveforms of game engine with clock of 50 MHz, touch and button inputs along with player scores and game states (5-7) as outputs. The states are 1 second apart from each other.

The instance of the game engine submodule was simulated using test benches. The inputs to the test bench are: a clock signal with a frequency of 50MHz, master reset signal that initializes the game at the start, touch screen and button values. The output signals of the test bench are: game states and the scores of the players. Based on the input the states change from main screen (1) to game mode screen (2), single player mode start screen (4) and then empty screen (5). After this state it takes 25x106 clock cycles to proceed to the next state. This is because the next state (ready screen) appears after a delay of 1 second. This is shown in Figure 7. The remaining state transition from empty screen (5), to ready screen (6) and so on are spaced apart with 1 second each as shown in Figure 8.

**5. HARDWARE IMPLEMENTATION AND TESTING**

**5.1. Single player Mode**

The single player mode of the game is controlled using push buttons, seven segment displays and touch screen input. Since the computer is the second player in this scenario, its input is considered as a time delay. The peripherals were connected in the manner as shown in Figure 9 to help manoeuvre through the game.

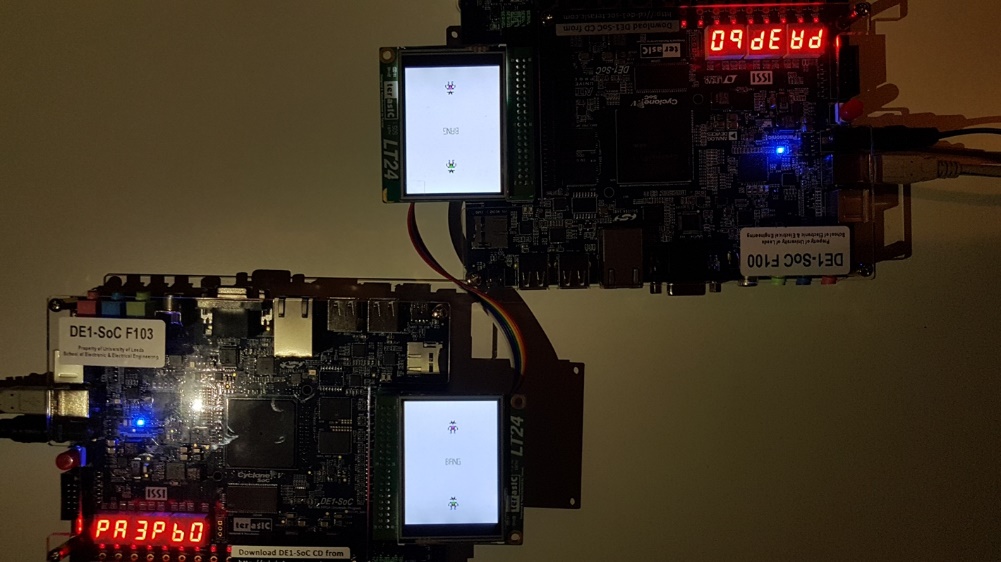
A close up of a sign

Description generated with very high confidence

**Figure 9:** Hardware implementation of the single player mode of the game using DE1-SOC board.

**5.2. Multiplayer Mode - Communication between two DE1-SoC Boards**

When connecting the 2nd DE1-SoC Board for the multiplayer mode, a method of communicating the inputs of the 2nd LCD touch screen to the main 1P board is required. This is achieved by using the Pin Planner to set pins on the empty GPIO-1 slot. This was determined by using the DE1-SoC GPIO Pin Map and Schematic documents (as shown in Appendix C – section 9). A total of 8 wires were connected across the two GPIO-1 sections. These wires included the touch signal from both LCD screens, the clock signal, the reset signal, three button signals and the ground reference. This allows the 2nd board to mimic the 1st board’s LCD screen so both players see the same gameplay. These simultaneous changes are due to the shared clock signal and this concept is reflected when resetting the overall game as well. There are two wires for the LCD screen because both boards need to know when either player has made an input i.e. taken a shot during main gameplay. There is only 1 wire to represent each of the remaining hardware components because the 1P board has control over selecting the gameplay modes.



**Figure 10:** Hardware implementation of the multi-player mode of the game using two DE1-SOC boards.

**6. CONCLUSIONS**

The project successfully demonstrates the development and implementation of the videogame Ready, Steady, Bang! The game can be played in either single player or multiplayer mode. The overall structure comprises of a series of complex modules that allows for simple and efficient debugging methods for detecting errors and shortcomings via the assistance of test benches through multiple simulations.

**7. REFERNCES**

1. Ready Steady Bang (standard edition). Smartphone [Game]. Animade: London, 2011.
2. Wikipedia. Linear-feedback shift register. [Online]. 2019. [Accessed 7 May 2019]. Available from: https://en.wikipedia.org/wiki/Linear-feedback\_shift\_register.
3. Electrical engineering and programming notepad. *16-bit colour generator (RGB565 colour picker)* [Online]. 2016. [Accessed 7 May 2019]. Available from: https://ee-programming-notepad.blogspot.com/2016/10/16-bit-color-generator-picker.html.

**APPENDIX – A**

**Appendix - A consists of all the Verilog module files used in the project.**

|  |
| --- |
| **SECTION 1:** N-bit counter module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Verilog Module Name : N\_Bit\_Counter    Code Author : Shrajan Bhandary  Date Created : 26/02/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used to count values ranging from 0 to required maximum value.  Once the maximum value is reached the counter is reset to 0. The counter  increments at every positive edge of clock signal and can be reset at every  positive edge of the reset signal i.e., when the reset button is pressed.  For this project the maximum value corresponds to the clock period (20 ms)  of the PWM Signal of the Servo Motor.    \*/  ////////////////////////////////////////////////////////////////////////////////  **module** N\_Bit\_Counter **#(** // Start of the module.    /\* Parameter List of the N\_Bit\_Counter \*/  **parameter** COUNTER\_VALUE\_WIDTH **=** 12**,** // The default width is 12 bits.  **parameter** COUNTER\_MAX\_VALUE **=** **(**2**\*\***COUNTER\_VALUE\_WIDTH**)-**1**,** // The maximum value that corresponds to (2^width - 1).  **parameter** COUNTER\_INCREMENT **=** 1 // The counter should increment by 1 every clock cycle.    **)(**  /\* Port List of the N\_Bit\_Counter \*/  **input** COUNTER\_CLOCK **,** // Counter increments according to the clock.  **input** COUNTER\_RESET **,** // Counter resets to 0 when reset becomes HIGH.  **input** COUNTER\_ENABLE **,** // Counter increments only if enable is HIGH.  **output** **reg** **[(**COUNTER\_VALUE\_WIDTH**-**1**):**0**]** COUNTER\_VALUE **=** 0 // The final count value of the counter.  **);**  /\* Local Parameter List of the N\_Bit\_Counter \*/  **localparam** ZERO **=** **{(**COUNTER\_VALUE\_WIDTH**){**1'b0**}};** // Local parameter with value 0 having default width of 12 bits.  **always** **@** **(** **posedge** COUNTER\_CLOCK **or** **posedge** COUNTER\_RESET **)** // Always statement such that the counter value changes when either  **begin** // reset or clock change from LOW to HIGH.    **if** **(** COUNTER\_RESET **)** // Check whether reset is HIGH.  **begin**  COUNTER\_VALUE **<=** ZERO**;** // Set counter value to 0 if reset is HIGH.  **end**    **else** **if** **(** COUNTER\_ENABLE **)** // Check if enable is HIGH  **begin**    **if** **(** COUNTER\_VALUE **>=** COUNTER\_MAX\_VALUE **)** // Check if counter value has surpassed maximum value.  **begin**  COUNTER\_VALUE **<=** ZERO**;** // Set counter value to 0 if counter value has surpassed maximum value.  **end**    **else**  **begin**  COUNTER\_VALUE **<=** COUNTER\_VALUE **+** COUNTER\_INCREMENT**;** // If none of the above conditions satisfy, then increment the counter value.  **end**  **end**  **end**  **endmodule** // End of the module. |

|  |
| --- |
| **SECTION 2:** N-bit comparator module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Verilog Module Name : N\_Bit\_Comparator    Code Author : Shrajan Bhandary  Date Created : 26/02/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used compare the value of two numbers. The output of the  module is HIGH when the first number is greater than or equal to the second  number. The output of the module is LOW when the first number is smaller  than the second number. The time period for which the output of the  comparator is HIGH determines the ON\_PERIOD of the PWM Signal of the Servo  Motor.    \*/  ////////////////////////////////////////////////////////////////////////////////  **module** N\_Bit\_Comparator **#(** // Start of the module.  /\* Parameter List of the N\_Bit\_Comparator \*/  **parameter** NUMBER\_WIDTH **=** 12 // The default width is 12 bits to match value from the counter.    **)(**  /\* Port List of the N\_Bit\_Comparator \*/  **input** **[(**NUMBER\_WIDTH**-**1**):**0**]** FIRST\_NUMBER **,** // The first number of the comparator.  **input** **[(**NUMBER\_WIDTH**-**1**):**0**]** SECOND\_NUMBER **,** // The second number of the comparator.  **output** **reg** FN\_GREATER\_THAN\_SN // The output of the comparator.  **);**  **always** **@** **(** FIRST\_NUMBER **,** SECOND\_NUMBER **)**  **begin**    **if** **(** FIRST\_NUMBER **>=** SECOND\_NUMBER **)** // LOW if second number is greater than first number, else HIGH.  **begin**  FN\_GREATER\_THAN\_SN **<=** 1'b1**;**  **end**    **else**  **begin**  FN\_GREATER\_THAN\_SN **<=** 1'b0**;**  **end**  **end**  **endmodule** // End of the module. |
| **SECTION 3:** Random Number generator |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Random\_Number  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 19/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used to obtain a random number that varies from minimum number  to maximum value.    \*/  ////////////////////////////////////////////////////////////////////////////////  /\* ceil(log2(N)) Preprocessor Macro \*/  `define clog2**(**x**)** **(** \  **((**x**)** **<=** 2**)** **?** 1 **:** \  **((**x**)** **<=** 4**)** **?** 2 **:** \  **((**x**)** **<=** 8**)** **?** 3 **:** \  **((**x**)** **<=** 16**)** **?** 4 **:** \  **((**x**)** **<=** 32**)** **?** 5 **:** \  **((**x**)** **<=** 64**)** **?** 6 **:** \  **((**x**)** **<=** 128**)** **?** 7 **:** \  **((**x**)** **<=** 256**)** **?** 8 **:** \  **((**x**)** **<=** 512**)** **?** 9 **:** \  **((**x**)** **<=** 1024**)** **?** 10 **:** \  **((**x**)** **<=** 2048**)** **?** 11 **:** \  **((**x**)** **<=** 4096**)** **?** 12 **:** 16**)**  **module** Random\_Number **#(** // Start of the module.  /\* Parameter List of the Random\_Number \*/  **parameter** MAXIMUM\_RANDOM\_NUMBER **=** 10 **,** // The maximum limit of random number.  **parameter** MINIMUM\_RANDOM\_NUMBER **=** 2 **,** // The minimum limit of random number.  **parameter** ADDRESS\_WIDTH **=** `clog2**(**MAXIMUM\_RANDOM\_NUMBER**)** // Determine the required width to denote the random number.  **)(**  /\* Port List of the Random\_Number \*/  **input** clock\_in**,** // The incoming clock is connected to this port.  **input** reset\_in**,** // The reset pin is connected to this port.  **input** random\_selector**,** // Select the value of random value.  **output** **reg** **[(**ADDRESS\_WIDTH**-**1**):**0**]**random\_number // The output that triggers the LED.  **);**  **reg** **[(**ADDRESS\_WIDTH**-**1**):**0**]**random\_counter**;** // Register that holds value of the random counter.    **always** **@** **(** **posedge** clock\_in **or** **posedge** reset\_in **)**  **begin**    **if** **(** reset\_in **)**  **begin**  random\_counter **<=** 0**;**  **end**    **else**  **begin**  random\_counter **<=** random\_counter **+** 1**;**  **end**  **end**    **always** **@** **(** **posedge** reset\_in **or** **posedge** random\_selector **)**  **begin**    **if** **(** reset\_in **)**  **begin**  random\_number **<=** 0**;**  **end**    **else** **if** **(** random\_counter **<** MINIMUM\_RANDOM\_NUMBER **)**  **begin**  random\_number **<=** random\_counter **+** MINIMUM\_RANDOM\_NUMBER**;**  **end**    **else** **if** **(** random\_counter **>** MAXIMUM\_RANDOM\_NUMBER **)**  **begin**  random\_number **<=** MAXIMUM\_RANDOM\_NUMBER**;**  **end**    **else**  **begin**  random\_number **<=** random\_counter**;**  **end**  **end**    **endmodule** |

|  |
| --- |
| **SECTION 4:** Random Number generator Test Bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Random\_Number\_tb  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 19/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used to obtain a random number that varies from minimum number  to maximum value.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns **/**100 ps  **module** Random\_Number\_tb**;**  /\* ceil(log2(N)) Preprocessor Macro \*/  `define clog2**(**x**)** **(** \  **((**x**)** **<=** 2**)** **?** 1 **:** \  **((**x**)** **<=** 4**)** **?** 2 **:** \  **((**x**)** **<=** 8**)** **?** 3 **:** \  **((**x**)** **<=** 16**)** **?** 4 **:** \  **((**x**)** **<=** 32**)** **?** 5 **:** \  **((**x**)** **<=** 64**)** **?** 6 **:** \  **((**x**)** **<=** 128**)** **?** 7 **:** \  **((**x**)** **<=** 256**)** **?** 8 **:** \  **((**x**)** **<=** 512**)** **?** 9 **:** \  **((**x**)** **<=** 1024**)** **?** 10 **:** \  **((**x**)** **<=** 2048**)** **?** 11 **:** \  **((**x**)** **<=** 4096**)** **?** 12 **:** 16**)**  /\* Parameter List of the Random\_Number\_tb \*/  **localparam** CLOCK\_FREQ **=** 100000 **;** // Current Clock frequency (in Hz).  **localparam** RST\_CYCLES **=** 1 **;** // Number of cycles of reset at beginning.  **localparam** WAIT\_PERIOD **=** 2 **;** // Creating a repeat block to make the reset the load signal.  **localparam** MAXIMUM\_RANDOM\_NUMBER **=** 10 **;** // The maximum limit of random number.  **localparam** MINIMUM\_RANDOM\_NUMBER **=** 2 **;** // The minimum limit of random number.  **localparam** ADDRESS\_WIDTH **=** `clog2**(**MAXIMUM\_RANDOM\_NUMBER**);** // Determine the required width to denote the random number.  /\* Test Bench Generated Signals of the Random\_Number\_tb \*/  **reg** TB\_CLOCK **;** // Connects to the clock of the random number generator  **reg** TB\_RESET **;** // Connects to the reset of the random number generator  **reg** TB\_SELECTOR **;** // Connects to the selector of the random number generator    /\* Device Under Test (DUT) Output Signals of the Random\_Number\_tb \*/  **wire** **[(**ADDRESS\_WIDTH**-**1**):**0**]** TB\_RANDOM\_NUMBER**;** // Connects to the random number value.  /\* Device Under Test (DUT) of the Random\_Number\_tb \*/  Random\_Number Random\_Number\_tb **(** // Setting the connections to their corresponding ports.  **.**clock\_in **(** TB\_CLOCK **),**  **.**reset\_in **(** TB\_RESET **),**  **.**random\_selector **(** TB\_SELECTOR **),**  **.**random\_number **(** TB\_RANDOM\_NUMBER **)**  **);**  /\* Reset the entire control system so that the servo initializes to the default value. \*/  **initial** **begin**  TB\_RESET **=** 1'b1**;** // Set the reset signal to HIGH.  **repeat(** RST\_CYCLES **)** **@** **(** **posedge** TB\_CLOCK **);** // Wait for a couple of clocks.  TB\_RESET **=** 1'b0**;** // Set the reset signal to LOW.  **end**  /\* Clock generator and simulation time limit. \*/  **initial** **begin**  TB\_CLOCK **=** 1'b0**;** // Initialise the clock to zero.  **end**  **real** HALF\_CLOCK\_PERIOD **=** **(**1000000000.0 **/** $itor**(**CLOCK\_FREQ**))** **/** 2.0**;** // Calculating the time delay for each half of the clock cycle and storing it in a variable.  **integer** half\_cycles **=** 0**;** // Variable to count the elapsed number of half cycles.  **always** **begin**    **repeat** **(** 5 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 87 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 52 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 63 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 90 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 42 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    TB\_SELECTOR **=** 1'b1**;** // Assign a value of 0 for the random selector.  **#**1**;** // Delay for 1 cycle.  TB\_SELECTOR **=** 1'b0**;** // Assign a value of 1 for the random selector.  **#**1**;** // Delay for 1 cycle.    **repeat** **(** 35 **)** // Repeat the loop for some time.  **begin**    TB\_SELECTOR **=** 1'b0**;** // Assign a value of 0 for the random selector.    /\* Generating individual half cycles of clock \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK **=** **~** TB\_CLOCK**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.    **end**    $stop**;** // Break the simulation  **end**  **endmodule** |
| **SECTION 5:** Frequency divider circuit module |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Frequency\_Divider  Target Device : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used to reduce the incoming clock rate to a fixed value.  The fixed value is 128 kHz and the incoming clock rate can vary from 128 kHz  to 100 MHz.    \*/  ////////////////////////////////////////////////////////////////////////////////  **module** Frequency\_Divider **#(** // Start of the module.  /\* Parameter List of the Frequency\_Divider \*/  **parameter** INCOMING\_CLOCK\_FREQUENCY **=** 50000000**,** // The minimum operable frequency is 128 kHz and the maximum operable frequency is 100 MHz.  **parameter** FIXED\_CLOCK\_FREQUENCY **=** 10**,** // All the servo default parameters are calculated at this frequency.  **parameter** TOGGLE **=** **(**INCOMING\_CLOCK\_FREQUENCY **/** FIXED\_CLOCK\_FREQUENCY**)/**2**,** // The toggle value is used to change the state of the output. The maximum value is 781.  **parameter** MAXIMUM\_WIDTH **=** 32 **,** // 10 bits to encompass all the possible values of toggle.  **parameter** INCREMENT **=** 1 // The value by which the clock counter should increase.    **)(**  /\* Port List of the Servo\_Motor\_Controller \*/  **input** FD\_CLOCK\_IN**,** // The incoming clock is connected to this port.  **input** FD\_RESET**,** // The reset pin is connected to this port.  **output** **reg** FD\_CLOCK\_OUT // This provides the fixed clock rate depending upon the divider value.  **);**  **wire** **[(**MAXIMUM\_WIDTH**-**1**):**0**]** CURRENT\_COUNTER **;** // This maximum value of the counter will be equal to the toggle which needs 10 bits.    **localparam** LOW **=** 0**;** // 1 bit Local parameter with value 0  **localparam** HIGH **=** 1**;** // 1 bit Local parameter with value 1  /\* Instantiating the N Bit counter to count values up to Toggle \*/  N\_Bit\_Counter **#** **(**  **.** COUNTER\_VALUE\_WIDTH **(** MAXIMUM\_WIDTH **),**  **.** COUNTER\_MAX\_VALUE **(** TOGGLE **),**  **.** COUNTER\_INCREMENT **(** INCREMENT **)**    **)** FD\_Toggler **(**  **.** COUNTER\_CLOCK **(** FD\_CLOCK\_IN **),**  **.** COUNTER\_RESET **(** FD\_RESET **),**  **.** COUNTER\_ENABLE **(** HIGH **),**  **.** COUNTER\_VALUE **(** CURRENT\_COUNTER **)**  **);**    **always** **@** **(** **posedge** FD\_CLOCK\_IN **or** **posedge** FD\_RESET **)** // Always statement such that the counter value changes when either  **begin** // reset or clock change from LOW to HIGH.    **if** **(** FD\_RESET **)** // Check whether reset is HIGH.  **begin**  FD\_CLOCK\_OUT **<=** LOW**;** // Reset the initialize the clock.  **end**    **else** **if** **(** CURRENT\_COUNTER **==** TOGGLE **-** 1 **)** // Check if the toggle value has been reached.  **begin**  FD\_CLOCK\_OUT **<=** **~** FD\_CLOCK\_OUT**;** // Switch (LOW to HIGH or HIGH to LOW ) the output clock when the current value has reached the toggle value.  **end**  **end**    **endmodule** |

|  |
| --- |
| **SECTION 6:** Frequency divider circuit module Test Bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : N - Channel Servo Motor Controller  Top level Entity Name : Frequency\_Divider\_tb  Target Device : Cyclone V    Code Author : Shrajan Bhandary  Date Created : 08/03/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is used to reduce the incoming clock rate to a fixed value.  The fixed value is 128 kHz and the incoming clock rate can vary from 128 kHz  to 100 MHz.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns **/**100 ps  **module** Frequency\_Divider\_tb**;**  /\* Parameter List of the Servo\_Motor\_Controller\_tb \*/  **localparam** NUM\_CYCLES **=** 10000000**;** // Simulate this many clock cycles. Maximum value is 1 billion.  **localparam** CLOCK\_FREQ **=** 50000000**;** // Current Clock frequency (in Hz).  **localparam** RST\_CYCLES **=** 2**;** // Number of cycles of reset at beginning.  /\* Test Bench Generated Signals of the Servo\_Motor\_Controller\_tb \*/  **reg** TB\_CLOCK\_IN **;** // Connects to the clock of the frequency divider circuit.  **reg** TB\_RESET **;** // Connects to the reset of the frequency divider circuit.    /\* Device Under Test (DUT) Output Signals of the Servo\_Motor\_Controller\_tb \*/  **wire** TB\_CLOCK\_OUT**;**  /\* Device Under Test (DUT) of the Servo\_Motor\_Controller\_tb \*/  Frequency\_Divider Frequency\_Divider\_DUT **(** // Setting the connections to their corresponding ports.  **.**FD\_CLOCK\_IN **(** TB\_CLOCK\_IN **),**  **.**FD\_RESET **(** TB\_RESET **),**  **.**FD\_CLOCK\_OUT **(** TB\_CLOCK\_OUT **)**  **);**  /\* Reset the entire control system so that the servo initializes to the default value. \*/  **initial** **begin**  TB\_RESET **=** 1'b1**;** // Set the reset signal to HIGH.  **repeat(** RST\_CYCLES **)** **@** **(** **posedge** TB\_CLOCK\_IN **);** // Wait for a couple of clocks.  TB\_RESET **=** 1'b0**;** // Set the reset signal to LOW.  **end**  /\* Clock generator and simulation time limit. \*/  **initial** **begin**  TB\_CLOCK\_IN **=** 1'b0**;** // Initialise the clock to zero.  **end**    **real** HALF\_CLOCK\_PERIOD **=** **(**1000000000.0 **/** $itor**(**CLOCK\_FREQ**))** **/** 2.0**;** // Calculating the time delay for each half of the clock cycle and storing it in a variable.  **integer** half\_cycles **=** 0**;** // Variable to count the elapsed number of half cycles.    **always** **begin**  /\* Generating individual half cycles of clock. \*/  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK\_IN **=** **~** TB\_CLOCK\_IN**;** // Toggle the clock signal.  half\_cycles **=** half\_cycles **+** 1**;** // Increment the count of number of half cycles.      /\* Check if we have simulated enough half clock cycles. \*/  **if** **(**half\_cycles **==** **(**2**\***NUM\_CYCLES**))**  **begin**  half\_cycles **=** 0**;** // Reset half cycles, so if we resume running with "run -all", we perform another chunk.  $stop**;** // Break the simulation  **end**  **end**  **endmodule** |

|  |
| --- |
| **SECTION 7:** Up Counter |
| /\*  \* N-Bit Up Counter  \* ----------------  \* By: Thomas Carpenter  \* Date: 13/03/2017  \*  \* Short Description  \* -----------------  \* This module is a simple up-counter with a count enable.  \* The counter has parameter controlled width, increment,  \* and maximum value.  \*  \*/  **module** UpCounterNbit **#(**  **parameter** WIDTH **=** 10**,** //10bit wide  **parameter** INCREMENT **=** 1**,** //Value to increment counter by each cycle  **parameter** MAX\_VALUE **=** **(**2**\*\***WIDTH**)-**1 //Maximum value default is 2^WIDTH - 1  **)(**  **input** clock**,**  **input** reset**,**  **input** enable**,** //Increments when enable is high  **output** **reg** **[(**WIDTH**-**1**):**0**]** countValue //Output is declared as "WIDTH" bits wide  **);**  **always** **@** **(posedge** clock**)** **begin**  **if** **(**reset**)** **begin**  //When reset is high, set back to 0  countValue **<=** **{(**WIDTH**){**1'b0**}};**  **end** **else** **if** **(**enable**)** **begin**  //Otherwise counter is not in reset  **if** **(**countValue **>=** MAX\_VALUE**[**WIDTH**-**1**:**0**])** **begin**  //If the counter value is equal or exceeds the maximum value  countValue **<=** **{(**WIDTH**){**1'b0**}};** //Reset back to 0  **end** **else** **begin**  //Otherwise increment  countValue **<=** countValue **+** INCREMENT**[**WIDTH**-**1**:**0**];**  **end**  **end**  **end**  **endmodule** |

|  |
| --- |
| **SECTION 8:** Seven Segment Controller |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Seven\_Segment\_Controller  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 20/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is to control the values being displayed on the seven segment  screens. Inverted output and inputs correspond to active low outputs and  inputs respectively.    \*/  ////////////////////////////////////////////////////////////////////////////////    **module** Seven\_Segment\_Controller **#(**  /\* Parameter List of the Seven\_Segment\_Controller \*/  **parameter** INVERT\_OUTPUT **=** 1 // Corresponds to active low or active high outputs.  **)(**    /\* Port List of the Seven\_Segment\_Controller \*/  **input** **[**3**:**0**]** num\_hex**,** // 4-bit hexadecimal number.  **output** **reg** **[**6**:**0**]** ss\_hex // Seven segment representation of the hexadecimal number.    **);**  **always** **@** **\***  **begin**    /\* For active low outputs. \*/  **if** **(** INVERT\_OUTPUT **)**  **begin**  **case(** num\_hex **)**  //0  4'b0000**:**ss\_hex **=** 7'b1000000**;**//7'b0111111;  //1  4'b0001**:**ss\_hex **=** 7'b1111001**;**//7'b0000110;  //2  4'b0010**:**ss\_hex **=** 7'b0100100**;**//7'b1011011;  //3  4'b0011**:**ss\_hex **=** 7'b0110000**;**//7'b1001111;  //4  4'b0100**:**ss\_hex **=** 7'b0011001**;**//7'b1100110;  //5  4'b0101**:**ss\_hex **=** 7'b0010010**;**//7'b1101101;  //6  4'b0110**:**ss\_hex **=** 7'b0000010**;**//7'b1111101;  //7  4'b0111**:**ss\_hex **=** 7'b1111000**;**//7'b0000111;  //8  4'b1000**:**ss\_hex **=** 7'b0000000**;**//7'b1111111;  //9  4'b1001**:**ss\_hex **=** 7'b0010000**;**//7'b1101111;  //A  4'b1010**:**ss\_hex **=** 7'b0001000**;**//7'b1110111;  //B  4'b1011**:**ss\_hex **=** 7'b0000011**;**//7'b1111100;  //C  4'b1100**:**ss\_hex **=** 7'b1000110**;**//7'b0111001;  //D  4'b1101**:**ss\_hex **=** 7'b0100001**;**//7'b1011110;  //E  4'b1110**:**ss\_hex **=** 7'b0000110**;**//7'b1111001;  //F  4'b1111**:**ss\_hex **=** 7'b0001100**;**//7'b1110001;  **endcase**  **end**    /\* For active high outputs. \*/  **else**  **begin**  **case(** num\_hex **)**  //0  4'b0000**:**ss\_hex **=** 7'b0111111**;**  //1  4'b0001**:**ss\_hex **=** 7'b0000110**;**  //2  4'b0010**:**ss\_hex **=** 7'b1011011**;**  //3  4'b0011**:**ss\_hex **=** 7'b1001111**;**  //4  4'b0100**:**ss\_hex **=** 7'b1100110**;**  //5  4'b0101**:**ss\_hex **=** 7'b1101101**;**  //6  4'b0110**:**ss\_hex **=** 7'b1111101**;**  //7  4'b0111**:**ss\_hex **=** 7'b0000111**;**  //8  4'b1000**:**ss\_hex **=** 7'b1111111**;**  //9  4'b1001**:**ss\_hex **=** 7'b1101111**;**  //A  4'b1010**:**ss\_hex **=** 7'b1110111**;**  //B  4'b1011**:**ss\_hex **=** 7'b1111100**;**  //C  4'b1100**:**ss\_hex **=** 7'b0111001**;**  //D  4'b1101**:**ss\_hex **=** 7'b1011110**;**  //E  4'b1110**:**ss\_hex **=** 7'b1111001**;**  //F  4'b1111**:**ss\_hex **=** 7'b1110011**;**  **endcase**  **end**  **end**  **endmodule** |

|  |
| --- |
| **SECTION 9:** Game Engine |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Game\_Engine  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 20/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is the main control panel of the game. All the control inputs and  screen states are manipulated to create game-play.    \*/  ////////////////////////////////////////////////////////////////////////////////  /\* ceil(log2(N)) Preprocessor Macro \*/  `define clog2**(**x**)** **(** \  **((**x**)** **<=** 2**)** **?** 1 **:** \  **((**x**)** **<=** 4**)** **?** 2 **:** \  **((**x**)** **<=** 8**)** **?** 3 **:** \  **((**x**)** **<=** 16**)** **?** 4 **:** \  **((**x**)** **<=** 32**)** **?** 5 **:** \  **((**x**)** **<=** 64**)** **?** 6 **:** \  **((**x**)** **<=** 128**)** **?** 7 **:** \  **((**x**)** **<=** 256**)** **?** 8 **:** \  **((**x**)** **<=** 512**)** **?** 9 **:** \  **((**x**)** **<=** 1024**)** **?** 10 **:** \  **((**x**)** **<=** 2048**)** **?** 11 **:** \  **((**x**)** **<=** 4096**)** **?** 12 **:** 16**)**  **module** Game\_Engine **#(** // Start of the module.  /\* Parameter List of the Game\_Engine \*/  **parameter** INCOMING\_CLOCK\_FREQUENCY **=** 50000000**,** // The frequency of the board is 50 MHz.  **parameter** REDUCED\_CLOCK\_FREQUENCY **=** 10 **,** // The fixed operable frequency is set to 2 Hz (Every clock cycle is 0.1 s long).    **parameter** NO\_GAME\_STATES **=** 35 **,** // The number of possible states (Screens).  **parameter** STATE\_ADDRESS\_WIDTH **=** `clog2**(**NO\_GAME\_STATES**),** // Number of bits required to store the different number of states.    **parameter** INVERTED\_INPUT **=** 1 **,** // Parameter to select between active LOW ( Invert = 1 ) inputs and active HIGH ( Invert = 0 )inputs.  **parameter** INVERTED\_OUTPUT **=** 1 **,** // Parameter to select between active LOW ( Invert = 1 ) outputs and active HIGH ( Invert = 0 )outputs.    **parameter** MAXIMUM\_BANG\_TRIGGER **=** 10 **,** // The maximum limit of time in seconds when BANG is triggered.  **parameter** MINIMUM\_BANG\_TRIGGER **=** 2 **,** // The minimum limit of time in seconds when BANG is triggered.  **parameter** MAXIMUM\_BANG\_COUNTER **=** MAXIMUM\_BANG\_TRIGGER **\*** REDUCED\_CLOCK\_FREQUENCY**,** // The maximum number of times the counter should count.  **parameter** MINIMUM\_BANG\_COUNTER **=** MINIMUM\_BANG\_TRIGGER **\*** REDUCED\_CLOCK\_FREQUENCY**,** // The maximum number of times the counter should count.  **parameter** BANG\_ADDRESS\_WIDTH **=** `clog2**(**MAXIMUM\_BANG\_COUNTER**),** // Determine the required width to of the BANG time.    **parameter** WINNING\_SCORE **=** 5 **,** // The game ends when either one of the player is the first to score the winning points.  **parameter** SCORE\_ADDRESS\_WIDTH **=** `clog2**(**WINNING\_SCORE**)** **,** // Determine the required width to of the scores.    **parameter** COUNTER\_INCREMENT **=** 1 **,** // The time gap in seconds between start screen, ready screen and steady screen.    **parameter** STATE\_TIME\_GAP **=** 1 **,** // The time gap in seconds between start screen, ready screen and steady screen.  **parameter** MAX\_GAP\_COUNTER **=** STATE\_TIME\_GAP **\*** REDUCED\_CLOCK\_FREQUENCY**,** // The maximum number of times the counter should count.  **parameter** GAP\_ADDRESS\_WIDTH **=** `clog2**(**MAX\_GAP\_COUNTER**)** // Determine the required width to of the gap time.  **)(**  /\* Port List of the Game\_Engine \*/    /\* Clock and Resets Inputs. \*/  **input** game\_clock\_in **,** // The incoming clock is connected to this port.  **input** game\_reset\_in **,** // The reset pin is connected to this port.    /\* Game-play control inputs. \*/  **input** game\_touch\_1\_button **,** // The touch input on screen 1.  **input** game\_touch\_2\_button **,** // The touch input on screen 2.  **input** game\_up\_button **,** // To switch between one player and two player version.  **input** game\_down\_button **,** // To switch between one player and two player version.  **input** game\_select\_button **,** // To select between one player or two player version.    **output** **reg** **[(**STATE\_ADDRESS\_WIDTH**-**1**):**0**]** display\_game\_state **,** // The register determines the current state of the game.    /\* Game-play Players' score output registers . \*/  **output** **reg** **[(**SCORE\_ADDRESS\_WIDTH**-**1**):**0**]** p\_1\_score **,** // Player 1 score.  **output** **reg** **[(**SCORE\_ADDRESS\_WIDTH**-**1**):**0**]** p\_2\_score // Player 2 score.  **);**    **wire** reduced\_clock**;** // Net that holds the reduced clock speed having reduced clock frequency .  **reg** counter\_reset **=** 1'b1**;** // Register that resets the counter of the gap calculator.    **reg** **[(**GAP\_ADDRESS\_WIDTH**-**1**):**0**]**current\_gap\_counter**;** // Register that holds value of counting.  **wire** gap\_elapsed**;** // Net that determines whether gap time has elapsed.    **reg** **[(**BANG\_ADDRESS\_WIDTH**-**1**):**0**]**current\_bang\_counter**;** // Register that holds the current value of bang counting.  **reg** **[(**BANG\_ADDRESS\_WIDTH**-**1**):**0**]**bang\_value**;** // Register that holds final value of bang counting.  **reg** bang\_counter\_reset **=** 1'b1**;** // Register that resets the counter of the bang calculator.  **wire** bang\_elapsed**;** // Register that determines whether bang time has elapsed.    **reg** **[(**BANG\_ADDRESS\_WIDTH**-**1**):**0**]**random\_number**;** // Register that holds value of the random number.    /\* Register list containing the active states of inputs of the Game\_Engine.\*/  **reg** active\_game\_reset\_in**;** // Inverts the input reset signal for active low inputs (if inverted input = 1).  **reg** active\_game\_touch\_1\_button**;** // Inverts the input touch 1 signal for active low inputs (if inverted input = 1).  **reg** active\_game\_touch\_2\_button**;** // Inverts the input touch 2 signal for active low inputs (if inverted input = 1).  **reg** active\_game\_up\_button**;** // Inverts the input up button signal for active low inputs (if inverted input = 1).  **reg** active\_game\_down\_button**;** // Inverts the input down button signal for active low inputs (if inverted input = 1).  **reg** active\_game\_select\_button**;** // Inverts the input select button signal for active low inputs (if inverted input = 1).      /\* Local Parameters list containing the screen states of LCD of the Game\_Engine.\*/  /\* States that determine the settings of the game.\*/  **localparam** A\_STATE **=** 6'b000001**;** // Main screen of the game.  **localparam** B\_STATE **=** 6'b000010**;** // One player version screen of the game.  **localparam** C\_STATE **=** 6'b000011**;** // Two player version screen of the game.    /\* States that control one player version of the game.\*/  **localparam** D\_1\_STATE **=** 6'b000100**;** // Start screen of the One player version of the game.  **localparam** E\_1\_STATE **=** 6'b000101**;** // Empty screen 1 of the One player version of the game.  **localparam** F\_1\_STATE **=** 6'b000110**;** // Ready screen of the One player version of the game.  **localparam** G\_1\_STATE **=** 6'b000111**;** // Empty screen 2 of the One player version of the game.  **localparam** H\_1\_STATE **=** 6'b001000**;** // Steady screen of the One player version of the game.  **localparam** I\_1\_STATE **=** 6'b001001**;** // Empty screen 3 of the One player version of the game.  **localparam** J\_1\_STATE **=** 6'b001010**;** // Bang screen of the One player version of the game.  **localparam** K\_1\_STATE **=** 6'b001011**;** // First Player kill screen of the One player version of the game.  **localparam** L\_1\_STATE **=** 6'b001100**;** // Second Player kill screen of the One player version of the game.  **localparam** M\_1\_STATE **=** 6'b001101**;** // Both Player kill screen of the One player version of the game.  **localparam** N\_1\_STATE **=** 6'b001110**;** // Next - First Player kill screen of the One player version of the game.  **localparam** O\_1\_STATE **=** 6'b001111**;** // Next - Second Player kill screen of the One player version of the game.  **localparam** P\_1\_STATE **=** 6'b010000**;** // Next - Both Player kill screen of the One player version of the game.  **localparam** Q\_1\_STATE **=** 6'b010001**;** // Player one Winner screen of the One player version of the game.  **localparam** R\_1\_STATE **=** 6'b010010**;** // Player two Winner screen of the One player version of the game.    /\* States that control two player version of the game.\*/  **localparam** D\_2\_STATE **=** 6'b010100**;** // Start screen of the Two player version of the game.  **localparam** E\_2\_STATE **=** 6'b010101**;** // Empty screen 1 of the Two player version of the game.  **localparam** F\_2\_STATE **=** 6'b010110**;** // Ready screen of the Two player version of the game.  **localparam** G\_2\_STATE **=** 6'b010111**;** // Empty screen 2 of the Two player version of the game.  **localparam** H\_2\_STATE **=** 6'b011000**;** // Steady screen of the Two player version of the game.  **localparam** I\_2\_STATE **=** 6'b011001**;** // Empty screen 3 of the Two player version of the game.  **localparam** J\_2\_STATE **=** 6'b011010**;** // Bang screen of the Two player version of the game.  **localparam** K\_2\_STATE **=** 6'b011011**;** // First Player kill screen of the Two player version of the game.  **localparam** L\_2\_STATE **=** 6'b011100**;** // Second Player kill screen of the Two player version of the game.  **localparam** M\_2\_STATE **=** 6'b011101**;** // Both Player kill screen of the Two player version of the game.  **localparam** N\_2\_STATE **=** 6'b011110**;** // Next - First Player kill screen of the Two player version of the game.  **localparam** O\_2\_STATE **=** 6'b011111**;** // Next - Second Player kill screen of the Two player version of the game.  **localparam** P\_2\_STATE **=** 6'b100000**;** // Next - Both Player kill screen of the Two player version of the game.  **localparam** Q\_2\_STATE **=** 6'b100001**;** // Player one Winner screen of the Two player version of the game.  **localparam** R\_2\_STATE **=** 6'b100010**;** // Player two Winner screen of the Two player version of the game.    /\* Game state-machine register. \*/  **reg** **[(**STATE\_ADDRESS\_WIDTH**-**1**):**0**]** game\_state **=** A\_STATE**;**    /\* Check the different possible game states and control inputs and determine the next states . \*/  **always** **@(** **posedge** game\_clock\_in **or** **posedge** active\_game\_reset\_in **)**  **begin**    **if** **(** active\_game\_reset\_in **)** // Check if the game reset button is pressed.  **begin**  game\_state **<=** A\_STATE**;** // Main screen of the game is the default page and the reset page .    /\* Reset the scores when new game starts. \*/  p\_1\_score **<=** 0**;**  p\_2\_score **<=** 0**;**  **end**    **else** // Other wise check the states of the game.  **begin**    **case** **(** game\_state **)** // Check current game state and take necessary actions.    ////////////////////////////////// A - STATE /////////////////////////////////  A\_STATE**:** **begin** // Current state is Main screen of the game.  **if** **(** active\_game\_touch\_1\_button **)** // Check if touch screen 1 is pressed.  **begin**  game\_state **<=** B\_STATE**;** // Go to One player version screen of the game.  **end**  **else** // Otherwise stay on same state.  **begin**  game\_state **<=** A\_STATE**;**  **end**  **end**    ////////////////////////////////// B - STATE /////////////////////////////////  B\_STATE**:** **begin** // Current state is One player version of the game.  **if** **(** active\_game\_select\_button **)** // Check if select button is pressed.  **begin**  game\_state **<=** D\_1\_STATE**;** // Go to Start screen of the game.  **end**    **else** **if** **(** active\_game\_down\_button **)** // Check if down button is pressed.  **begin**  game\_state **<=** C\_STATE**;** // Go to Two player version screen of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** B\_STATE**;**  **end**  **end**    ////////////////////////////////// C - STATE /////////////////////////////////  C\_STATE**:** **begin** // Current state is Two player version of the game.  **if** **(** active\_game\_select\_button **)** // Check if select button is pressed.  **begin**  game\_state **<=** D\_2\_STATE**;** // Go to Start screen of the game.  **end**    **else** **if** **(** active\_game\_up\_button **)** // Check if up button is pressed.  **begin**  game\_state **<=** B\_STATE**;** // Go to One player version screen of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** C\_STATE**;**  **end**  **end**    /////////////////////////////////////////////////////////////////////////////////////////////////////////  /////////////////////////////////// ONE PLAYER VERSION //////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////////////////////////////////    ////////////////////////////////// D 1 - STATE /////////////////////////////////  D\_1\_STATE**:** **begin** // Current state is Start screen of the one player version of the game.  **if** **(** active\_game\_touch\_1\_button **)** // Check if touch screen 1 is pressed.  **begin**  game\_state **<=** E\_1\_STATE**;** // Go to Empty screen 1 of the one player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** D\_1\_STATE**;**  **end**  **end**    ////////////////////////////////// E 1 - STATE /////////////////////////////////  E\_1\_STATE**:** **begin** // Current state is Empty screen 1 of the one player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** F\_1\_STATE**;** // Go to Ready screen of the one player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** E\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// F 1 - STATE /////////////////////////////////  F\_1\_STATE**:** **begin** // Current state is Ready screen of the one player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** G\_1\_STATE**;** // Go to Empty screen 2 of the one player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** F\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// G 1 - STATE /////////////////////////////////  G\_1\_STATE**:** **begin** // Current state is Empty screen 2 of the one player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** H\_1\_STATE**;** // Go to Steady screen of the one player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** G\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// H 1 - STATE /////////////////////////////////  H\_1\_STATE**:** **begin** // Current state is Steady screen of the one player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** I\_1\_STATE**;** // Go to Empty screen 3 of the one player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** H\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// I 1 - STATE /////////////////////////////////  I\_1\_STATE**:** **begin** // Current state is Empty screen 3 of the one player version of the game.  **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen pre-maturely.  **begin**  game\_state **<=** L\_1\_STATE**;** // Go to Second Player kill screen of the one player version of the game.  bang\_counter\_reset **<=** 1'b1**;**  p\_2\_score **<=** p\_2\_score **+** 1**;**  **end**    **else** **if** **(** bang\_elapsed **)** // Check if bang time has elapsed.  **begin**  game\_state **<=** J\_1\_STATE**;** // Go to Bang screen of the one player version of the game.  bang\_counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** I\_1\_STATE**;**  bang\_counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// J 1 - STATE /////////////////////////////////  J\_1\_STATE**:** **begin** // Current state is Bang screen of the one player version of the game.    /\* Check if both the player and computer tap at the same time . \*/  **if** **(** active\_game\_touch\_1\_button **&&** gap\_elapsed **)**  **begin**  game\_state **<=** M\_1\_STATE**;** // Go to Both Player kill screen of the one player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen.  **begin**  game\_state **<=** K\_1\_STATE**;** // Go to First Player kill screen of the one player version of the game.  counter\_reset **<=** 1'b1**;**  p\_1\_score **<=** p\_1\_score **+** 1**;**  **end**    **else** **if** **(** gap\_elapsed **)** // Check if computer time has elapsed.  **begin**  game\_state **<=** L\_1\_STATE**;** // Go to Second Player kill screen of the one player version of the game.  counter\_reset **<=** 1'b1**;**  p\_2\_score **<=** p\_2\_score **+** 1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** J\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// K 1 - STATE /////////////////////////////////  K\_1\_STATE**:** **begin** // Current state is First Player kill screen of the one player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** N\_1\_STATE**;** // Go to Next - First Player kill screen of the one player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** K\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// L 1 - STATE /////////////////////////////////  L\_1\_STATE**:** **begin** // Current state is Second Player kill screen of the one player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** O\_1\_STATE**;** // Go to Next - Second Player kill screen of the one player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** L\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// M 1 - STATE /////////////////////////////////  M\_1\_STATE**:** **begin** // Current state is Both Players kill screen of the one player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** P\_1\_STATE**;** // Go to Next - Both Players kill screen of the one player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** M\_1\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// N 1 - STATE /////////////////////////////////  N\_1\_STATE**:** **begin** // Current state is Next - First Player kill screen of the one player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**    /\* Determine whether any player has reached the winning score. \*/  **if** **(** p\_1\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** Q\_1\_STATE**;** // Go to Player one Winner screen of the one player version of the game.  **end**  **else** **if** **(** p\_2\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** R\_1\_STATE**;** // Go to Player two Winner screen of the one player version of the game.  **end**  **else**  **begin**  game\_state **<=** E\_1\_STATE**;** // Otherwise go to Empty Screen 1 of the one player version of the game.  **end**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** N\_1\_STATE**;**  **end**  **end**    ////////////////////////////////// O 1 - STATE /////////////////////////////////  O\_1\_STATE**:** **begin** // Current state is Next - Second Player kill screen of the one player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**    /\* Determine whether any player has reached the winning score. \*/  **if** **(** p\_1\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** Q\_1\_STATE**;** // Go to Player one Winner screen of the one player version of the game.  **end**  **else** **if** **(** p\_2\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** R\_1\_STATE**;** // Go to Player two Winner screen of the one player version of the game.  **end**  **else**  **begin**  game\_state **<=** E\_1\_STATE**;** // Otherwise go to Empty Screen 1 of the one player version of the game.  **end**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** O\_1\_STATE**;**  **end**  **end**    ////////////////////////////////// P 1 - STATE /////////////////////////////////  P\_1\_STATE**:** **begin** // Current state is Both Players kill screen of the one player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**  game\_state **<=** E\_1\_STATE**;** // Otherwise go to Empty Screen 1 of the one player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** P\_1\_STATE**;**  **end**  **end**    ////////////////////////////////// Q 1 - STATE /////////////////////////////////  Q\_1\_STATE**:** game\_state **<=** Q\_1\_STATE**;** // Stay on same state. Current state is First Player winner screen of the one player version of the game.    ////////////////////////////////// R 1 - STATE /////////////////////////////////  R\_1\_STATE**:** game\_state **<=** R\_1\_STATE**;** // Stay on same state. Current state is Second Player winner screen of the one player version of the game.      /////////////////////////////////////////////////////////////////////////////////////////////////////////  /////////////////////////////////// TWO PLAYER VERSION //////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////////////////////////////////    ////////////////////////////////// D 2 - STATE /////////////////////////////////  D\_2\_STATE**:** **begin** // Current state is Start screen of the two player version of the game.  **if** **(** active\_game\_touch\_1\_button **)** // Check if touch screen 1 is pressed.  **begin**  game\_state **<=** E\_2\_STATE**;** // Go to Empty screen 1 of the two player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** D\_2\_STATE**;**  **end**  **end**    ////////////////////////////////// E 2 - STATE /////////////////////////////////  E\_2\_STATE**:** **begin** // Current state is Empty screen 1 of the two player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** F\_2\_STATE**;** // Go to Ready screen of the two player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** E\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// F 2 - STATE /////////////////////////////////  F\_2\_STATE**:** **begin** // Current state is Ready screen of the two player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** G\_2\_STATE**;** // Go to Empty screen 2 of the two player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** F\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// G 2 - STATE /////////////////////////////////  G\_2\_STATE**:** **begin** // Current state is Empty screen 2 of the two player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** H\_2\_STATE**;** // Go to Steady screen of the two player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** G\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// H 2 - STATE /////////////////////////////////  H\_2\_STATE**:** **begin** // Current state is Steady screen of the two player version of the game.  **if** **(** gap\_elapsed **)** // Check if time has elapsed.  **begin**  game\_state **<=** I\_2\_STATE**;** // Go to Empty screen 3 of the two player version of the game.  counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** H\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// I 2 - STATE /////////////////////////////////  I\_2\_STATE**:** **begin** // Current state is Empty screen 3 of the two player version of the game.  **if** **(** active\_game\_touch\_1\_button **)** // Check if the player one has tapped the screen pre-maturely.  **begin**  game\_state **<=** L\_2\_STATE**;** // Go to Second Player kill screen of the two player version of the game.  bang\_counter\_reset **<=** 1'b1**;**  p\_2\_score **<=** p\_2\_score **+** 1**;**  **end**    **else** **if** **(** active\_game\_touch\_2\_button **)** // Check if the player two has tapped the screen pre-maturely.  **begin**  game\_state **<=** K\_2\_STATE**;** // Go to First Player kill screen of the two player version of the game.  bang\_counter\_reset **<=** 1'b1**;**  p\_1\_score **<=** p\_1\_score **+** 1**;**  **end**    **else** **if** **(** bang\_elapsed **)** // Check if bang time has elapsed.  **begin**  game\_state **<=** J\_2\_STATE**;** // Go to Bang screen of the two player version of the game.  bang\_counter\_reset **<=** 1'b1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** I\_2\_STATE**;**  bang\_counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// J 2 - STATE /////////////////////////////////  J\_2\_STATE**:** **begin** // Current state is Bang screen of the two player version of the game.    /\* Check if both the players tap at the same time . \*/  **if** **(** active\_game\_touch\_1\_button **&&** active\_game\_touch\_2\_button **)**  **begin**  game\_state **<=** M\_2\_STATE**;** // Go to Both Player kill screen of the two player version of the game.  **end**    **else** **if** **(** active\_game\_touch\_1\_button **)** // Check if the player one has tapped the screen.  **begin**  game\_state **<=** K\_2\_STATE**;** // Go to First Player kill screen of the two player version of the game.  p\_1\_score **<=** p\_1\_score **+** 1**;**  **end**    **else** **if** **(** active\_game\_touch\_2\_button **)** // Check if the player two has tapped the screen.  **begin**  game\_state **<=** L\_2\_STATE**;** // Go to Second Player kill screen of the two player version of the game.  p\_2\_score **<=** p\_2\_score **+** 1**;**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** J\_2\_STATE**;**  **end**  **end**    ////////////////////////////////// K 2 - STATE /////////////////////////////////  K\_2\_STATE**:** **begin** // Current state is First Player kill screen of the two player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** N\_2\_STATE**;** // Go to Next - First Player kill screen of the two player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** K\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// L 2 - STATE /////////////////////////////////  L\_2\_STATE**:** **begin** // Current state is Second Player kill screen of the two player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** O\_2\_STATE**;** // Go to Next - Second Player kill screen of the two player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** L\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// M 2 - STATE /////////////////////////////////  M\_2\_STATE**:** **begin** // Current state is Both Players kill screen of the two player version of the game.    **if** **(** gap\_elapsed **)** // Check if certain period of time as elapsed.  **begin**  counter\_reset **<=** 1'b1**;**  game\_state **<=** P\_2\_STATE**;** // Go to Next - Both Players kill screen of the two player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** M\_2\_STATE**;**  counter\_reset **<=** 1'b0**;**  **end**  **end**    ////////////////////////////////// N 2 - STATE /////////////////////////////////  N\_2\_STATE**:** **begin** // Current state is Next - First Player kill screen of the two player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**    /\* Determine whether any player has reached the winning score. \*/  **if** **(** p\_1\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** Q\_2\_STATE**;** // Go to Player two Winner screen of the two player version of the game.  **end**  **else** **if** **(** p\_2\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** R\_2\_STATE**;** // Go to Player two Winner screen of the two player version of the game.  **end**  **else**  **begin**  game\_state **<=** E\_2\_STATE**;** // Otherwise go to Empty Screen 2 of the two player version of the game.  **end**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** N\_2\_STATE**;**  **end**  **end**    ////////////////////////////////// O 2 - STATE /////////////////////////////////  O\_2\_STATE**:** **begin** // Current state is Next - Second Player kill screen of the two player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**    /\* Determine whether any player has reached the winning score. \*/  **if** **(** p\_1\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** Q\_2\_STATE**;** // Go to Player two Winner screen of the two player version of the game.  **end**  **else** **if** **(** p\_2\_score **>=** WINNING\_SCORE **)**  **begin**  game\_state **<=** R\_2\_STATE**;** // Go to Player two Winner screen of the two player version of the game.  **end**  **else**  **begin**  game\_state **<=** E\_2\_STATE**;** // Otherwise go to Empty Screen 2 of the two player version of the game.  **end**  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** O\_2\_STATE**;**  **end**  **end**    ////////////////////////////////// P 2 - STATE /////////////////////////////////  P\_2\_STATE**:** **begin** // Current state is Both Players kill screen of the two player version of the game.    **if** **(** active\_game\_touch\_1\_button **)** // Check if the player has tapped the screen to proceed to the next screen.  **begin**  game\_state **<=** E\_2\_STATE**;** // Otherwise go to Empty Screen 1 of the two player version of the game.  **end**    **else** // Otherwise stay on same state.  **begin**  game\_state **<=** P\_2\_STATE**;**  **end**  **end**    ////////////////////////////////// Q 2 - STATE /////////////////////////////////  Q\_2\_STATE**:** game\_state **<=** Q\_2\_STATE**;** // Stay on same state. Current state is First Player winner screen of the two player version of the game.    ////////////////////////////////// R 2 - STATE /////////////////////////////////  R\_2\_STATE**:** game\_state **<=** R\_2\_STATE**;** // Stay on same state. Current state is Second Player winner screen of the two player version of the game.    **default:** game\_state **<=** A\_STATE**;**  **endcase**  **end**  **end**    /\* Display the correct output on the LCD based on the game state. \*/  **always** **@(** game\_state **)** // Check current game state and take necessary actions by displaying corresponding screens.  **begin**    **case** **(** game\_state **)**    A\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    B\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    C\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    D\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    E\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    F\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    G\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    H\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    I\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    J\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    K\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    L\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    M\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    N\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    O\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**  P\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    Q\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    R\_1\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    /\* Two player states. \*/    D\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    E\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    F\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    G\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    H\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    I\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    J\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    K\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    L\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    M\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    N\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    O\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**  P\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    Q\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**    R\_2\_STATE**:** **begin**  display\_game\_state **<=** game\_state **;**  **end**      **default** **:** **begin**  display\_game\_state **<=** game\_state **;**  **end**      **endcase**  **end**  /\* To check whether the inputs are active LOW or not and then take necessary actions \*/  **always** **@** **(** **posedge** game\_clock\_in **)**  **begin** // Always statement that changes with input signals.    **if** **(** INVERTED\_INPUT **)** // Check if the inputs are active LOW or active HIGH.  **begin** // If yes, invert the inputs.  active\_game\_reset\_in **<=** **~** game\_reset\_in**;**  active\_game\_touch\_1\_button **<=** **~** game\_touch\_1\_button**;**  active\_game\_touch\_2\_button **<=** **~** game\_touch\_2\_button**;**  active\_game\_up\_button **<=** **~** game\_up\_button**;**  active\_game\_down\_button **<=** **~** game\_down\_button**;**  active\_game\_select\_button **<=** **~** game\_select\_button**;**  **end**    **else**  **begin** // If no, assign the same inputs.  active\_game\_reset\_in **<=** game\_reset\_in**;**  active\_game\_touch\_1\_button **<=** game\_touch\_1\_button**;**  active\_game\_touch\_2\_button **<=** game\_touch\_2\_button**;**  active\_game\_up\_button **<=** game\_up\_button**;**  active\_game\_down\_button **<=** game\_down\_button**;**  active\_game\_select\_button **<=** game\_select\_button**;**  **end**  **end**    /\* Instantiating the frequency generator to reduce the speed of the clock. \*/  Frequency\_Divider **#** **(**  **.** INCOMING\_CLOCK\_FREQUENCY **(** INCOMING\_CLOCK\_FREQUENCY **),** // Setting the parameter values.  **.** FIXED\_CLOCK\_FREQUENCY **(** REDUCED\_CLOCK\_FREQUENCY **)**    **)** FG\_BLOCK **(**  **.** FD\_CLOCK\_IN **(** game\_clock\_in **),** // Setting the connections to their corresponding ports.  **.** FD\_RESET **(** active\_game\_reset\_in **),**  **.** FD\_CLOCK\_OUT **(** reduced\_clock **)**  **);**    /////////////////////////////////////////////////////////////////////////////////////////////////////////  /////////////////////////////////// GAP CALCULATOR ///////////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////////////////////////////////    **always** **@** **(** **posedge** reduced\_clock **or** **posedge** counter\_reset **)**  **begin**    **if** **(** counter\_reset **)**  **begin**  current\_gap\_counter **<=** 0**;**  **end**    **else**  **begin**  current\_gap\_counter **<=** current\_gap\_counter **+** 1**;**  **end**  **end**    **assign** gap\_elapsed **=** **(** current\_gap\_counter **>=** MAX\_GAP\_COUNTER **);**    /////////////////////////////////////////////////////////////////////////////////////////////////////////  /////////////////////////////////// RANDOM GENERATOR //////////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////////////////////////////////    Random\_Number **#(**  **.**MAXIMUM\_RANDOM\_NUMBER **(** MAXIMUM\_BANG\_COUNTER **),**  **.**MINIMUM\_RANDOM\_NUMBER **(** MINIMUM\_BANG\_COUNTER **),**  **.**ADDRESS\_WIDTH **(** BANG\_ADDRESS\_WIDTH **)**    **)** RANDOM\_NUMBER\_GENERATOR **(**    **.**clock\_in **(** game\_clock\_in **),**  **.**reset\_in **(** active\_game\_reset\_in **),**  **.**random\_selector **(** active\_game\_touch\_1\_button **),**  **.**random\_number **(** bang\_value **)**  **);**    /////////////////////////////////////////////////////////////////////////////////////////////////////////  /////////////////////////////////// BANG CALCULATOR //////////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////////////////////////////////    **always** **@** **(** **posedge** reduced\_clock **or** **posedge** bang\_counter\_reset **)**  **begin**    **if** **(** bang\_counter\_reset **)**  **begin**  current\_bang\_counter **<=** 0**;**  **end**    **else**  **begin**  current\_bang\_counter **<=** current\_bang\_counter **+** 1**;**  **end**  **end**    **assign** bang\_elapsed **=** **(** current\_bang\_counter **>=** bang\_value **);**    **endmodule** // End of the module. |

|  |
| --- |
| **SECTION 10:** Game Engine Test Bench |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Game\_Engine\_tb  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 20/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is the main control panel of the game. All the control inputs and  screen states are manipulated to create game-play.    \*/  ////////////////////////////////////////////////////////////////////////////////  `timescale 1 ns **/**100 ps  **module** Game\_Engine\_tb**;**  /\* Parameter List of the Game\_Engine\_tb \*/  **localparam** NUM\_CYCLES **=** 1000000000**;** // Simulate this many clock cycles. Maximum value is 1 billion.  **localparam** CLOCK\_FREQ **=** 50000000**;** // Current Clock frequency (in Hz).  **localparam** RST\_CYCLES **=** 2**;** // Number of cycles of reset at beginning.  /\* Test Bench Generated Signals of the Game\_Engine\_tb \*/  **reg** TB\_CLOCK\_IN **;** // Connects to the clock of the frequency divider circuit.  **reg** TB\_RESET **;** // Connects to the reset of the frequency divider circuit.  **reg** TB\_TOUCH\_1\_BUTTON **;** // Connects to the touch input on screen 1.  **reg** TB\_TOUCH\_2\_BUTTON **;** // Connects to the touch input on screen 2.  **reg** TB\_UP\_BUTTON **;** // Connects to the button to switch between one player and two player version.  **reg** TB\_DOWN\_BUTTON **;** // Connects to the button to switch between one player and two player version.  **reg** TB\_SELECT\_BUTTON **;** // Connects to the button to select between one player or two player version..    /\* Device Under Test (DUT) Output Signals of the Game\_Engine\_tb \*/  **wire** **[**5**:**0**]** TB\_GAME\_STATE**;** // Connects to the register determines the current state of the game.  **wire** **[**2**:**0**]** TB\_P\_1\_SCORE **;** // Connects to Player 1 score.  **wire** **[**2**:**0**]** TB\_P\_2\_SCORE **;** // Connects to Player 2 score.  /\* Device Under Test (DUT) of the Game\_Engine\_tb \*/  Game\_Engine Game\_Engine\_tb\_DUT **(** // Setting the connections to their corresponding ports.  **.**game\_clock\_in **(** TB\_CLOCK\_IN **),**  **.**game\_reset\_in **(** TB\_RESET **),**  **.**game\_touch\_1\_button **(** TB\_TOUCH\_1\_BUTTON **),**  **.**game\_touch\_2\_button **(** TB\_TOUCH\_2\_BUTTON **),**  **.**game\_up\_button **(** TB\_UP\_BUTTON **),**  **.**game\_down\_button **(** TB\_DOWN\_BUTTON **),**  **.**game\_select\_button **(** TB\_SELECT\_BUTTON **),**  **.**display\_game\_state **(** TB\_GAME\_STATE **),**  **.**p\_1\_score **(** TB\_P\_1\_SCORE **),**  **.**p\_2\_score **(** TB\_P\_2\_SCORE **)**  **);**  /\* Reset the entire control system so that the servo initializes to the default value. \*/  **initial** **begin**  TB\_RESET **=** 1'b1**;** // Set the reset signal to HIGH.  **repeat(** RST\_CYCLES **)** **@** **(** **posedge** TB\_CLOCK\_IN **);** // Wait for a couple of clocks.  TB\_RESET **=** 1'b0**;** // Set the reset signal to LOW.  **end**  /\* Clock generator and simulation time limit. \*/  **initial** **begin**  TB\_CLOCK\_IN **=** 1'b0**;** // Initialise the clock to zero.  **end**    **real** HALF\_CLOCK\_PERIOD **=** **(**1000000000.0 **/** $itor**(**CLOCK\_FREQ**))** **/** 2.0**;** // Calculating the time delay for each half of the clock cycle and storing it in a variable.  **integer** half\_cycles **=** 0**;** // Variable to count the elapsed number of half cycles.    **always** **begin**  //Generate the next half cycle of clock  **#(**HALF\_CLOCK\_PERIOD**);** // Delay for half a clock period.  TB\_CLOCK\_IN **=** **~**TB\_CLOCK\_IN**;** // Toggle the clock  half\_cycles **=** half\_cycles **+** 1**;** // Increment the counter    TB\_TOUCH\_1\_BUTTON **=** 1'b1**;**  TB\_TOUCH\_2\_BUTTON **=** 1'b0**;**  TB\_UP\_BUTTON **=** 1'b0**;**  TB\_DOWN\_BUTTON **=** 1'b0**;**  TB\_SELECT\_BUTTON **=** 1'b1**;**    //Check if we have simulated enough half clock cycles  **if** **(**half\_cycles **==** **(**2**\***NUM\_CYCLES**))** **begin**  //Once the number of cycles has been reached  half\_cycles **=** 0**;** // Reset half cycles, so if we resume running with "run -all", we perform another chunk.  $stop**;** // Break the simulation.  **end**  **end**  **endmodule** |

|  |
| --- |
| **SECTION 12:** Video Engine |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Video\_Engine  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 20/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module defines the values of pixels for different states of the game.    \*/  ////////////////////////////////////////////////////////////////////////////////  /\* ceil(log2(N)) Preprocessor Macro \*/  `define clog2**(**x**)** **(** \  **((**x**)** **<=** 2**)** **?** 1 **:** \  **((**x**)** **<=** 4**)** **?** 2 **:** \  **((**x**)** **<=** 8**)** **?** 3 **:** \  **((**x**)** **<=** 16**)** **?** 4 **:** \  **((**x**)** **<=** 32**)** **?** 5 **:** \  **((**x**)** **<=** 64**)** **?** 6 **:** \  **((**x**)** **<=** 128**)** **?** 7 **:** \  **((**x**)** **<=** 256**)** **?** 8 **:** \  **((**x**)** **<=** 512**)** **?** 9 **:** \  **((**x**)** **<=** 1024**)** **?** 10 **:** \  **((**x**)** **<=** 2048**)** **?** 11 **:** \  **((**x**)** **<=** 4096**)** **?** 12 **:** 16**)**  **module** Video\_Engine **#(** // Start of the module.  /\* Parameter List of the Video\_Engine \*/  **parameter** LCD\_WIDTH **=** 240 **,** // The number of pixels in the x-direction is 240.  **parameter** LCD\_HEIGHT **=** 320 **,** // The number of pixels in the y-direction is 320.  **parameter** X\_ADDRESS\_WIDTH **=** `clog2**(**LCD\_WIDTH**)** **,** // Number of bits required to store the x-address.  **parameter** Y\_ADDRESS\_WIDTH **=** `clog2**(**LCD\_HEIGHT**)** **,** // Number of bits required to store the y-address.    **parameter** NO\_GAME\_STATES **=** 35 **,** // The number of possible states (Screens).  **parameter** STATE\_ADDRESS\_WIDTH **=** `clog2**(**NO\_GAME\_STATES**),** // Number of bits required to store the different number of states.    **parameter** PIXEL\_ADDRESS\_WIDTH **=** 16 **,** // Number of bits required to colour a pixel.  **parameter** PLAYER\_ID **=** 1 // The ID is either 1 or 2 depending on the board that the program runs.  **)(**  /\* Port List of the Video\_Engine \*/  **input** **[(**X\_ADDRESS\_WIDTH**-**1**):**0**]** lcd\_pixel\_x\_address **,** // The current x-coordinate of the LCD pixel.  **input** **[(**Y\_ADDRESS\_WIDTH**-**1**):**0**]** lcd\_pixel\_y\_address **,** // The current y-coordinate of the LCD pixel.  **input** **[(**STATE\_ADDRESS\_WIDTH**-**1**):**0**]** game\_states **,** // The current state of the game screen.  **output** **reg** **[(**PIXEL\_ADDRESS\_WIDTH**-**1**):**0**]** lcd\_pixel\_data // The data values that determine the colour of the pixel.  **);**  /\* Local Parameters list containing the screen states of LCD of the Game\_Engine.\*/  /\* States that determine the settings of the game.\*/  **localparam** A\_STATE **=** 6'b000001**;** // Main screen of the game.  **localparam** B\_STATE **=** 6'b000010**;** // One player version screen of the game.  **localparam** C\_STATE **=** 6'b000011**;** // Two player version screen of the game.    /\* States that control one player version of the game.\*/  **localparam** D\_1\_STATE **=** 6'b000100**;** // Start screen of the One player version of the game.  **localparam** E\_1\_STATE **=** 6'b000101**;** // Empty screen 1 of the One player version of the game.  **localparam** F\_1\_STATE **=** 6'b000110**;** // Ready screen of the One player version of the game.  **localparam** G\_1\_STATE **=** 6'b000111**;** // Empty screen 2 of the One player version of the game.  **localparam** H\_1\_STATE **=** 6'b001000**;** // Steady screen of the One player version of the game.  **localparam** I\_1\_STATE **=** 6'b001001**;** // Empty screen 3 of the One player version of the game.  **localparam** J\_1\_STATE **=** 6'b001010**;** // Bang screen of the One player version of the game.  **localparam** K\_1\_STATE **=** 6'b001011**;** // First Player kill screen of the One player version of the game.  **localparam** L\_1\_STATE **=** 6'b001100**;** // Second Player kill screen of the One player version of the game.  **localparam** M\_1\_STATE **=** 6'b001101**;** // Both Player kill screen of the One player version of the game.  **localparam** N\_1\_STATE **=** 6'b001110**;** // Next - First Player kill screen of the One player version of the game.  **localparam** O\_1\_STATE **=** 6'b001111**;** // Next - Second Player kill screen of the One player version of the game.  **localparam** P\_1\_STATE **=** 6'b010000**;** // Next - Both Player kill screen of the One player version of the game.  **localparam** Q\_1\_STATE **=** 6'b010001**;** // Player one Winner screen of the One player version of the game.  **localparam** R\_1\_STATE **=** 6'b010010**;** // Player two Winner screen of the One player version of the game.    /\* States that control two player version of the game.\*/  **localparam** D\_2\_STATE **=** 6'b010100**;** // Start screen of the Two player version of the game.  **localparam** E\_2\_STATE **=** 6'b010101**;** // Empty screen 1 of the Two player version of the game.  **localparam** F\_2\_STATE **=** 6'b010110**;** // Ready screen of the Two player version of the game.  **localparam** G\_2\_STATE **=** 6'b010111**;** // Empty screen 2 of the Two player version of the game.  **localparam** H\_2\_STATE **=** 6'b011000**;** // Steady screen of the Two player version of the game.  **localparam** I\_2\_STATE **=** 6'b011001**;** // Empty screen 3 of the Two player version of the game.  **localparam** J\_2\_STATE **=** 6'b011010**;** // Bang screen of the Two player version of the game.  **localparam** K\_2\_STATE **=** 6'b011011**;** // First Player kill screen of the Two player version of the game.  **localparam** L\_2\_STATE **=** 6'b011100**;** // Second Player kill screen of the Two player version of the game.  **localparam** M\_2\_STATE **=** 6'b011101**;** // Both Player kill screen of the Two player version of the game.  **localparam** N\_2\_STATE **=** 6'b011110**;** // Next - First Player kill screen of the Two player version of the game.  **localparam** O\_2\_STATE **=** 6'b011111**;** // Next - Second Player kill screen of the Two player version of the game.  **localparam** P\_2\_STATE **=** 6'b100000**;** // Next - Both Player kill screen of the Two player version of the game.  **localparam** Q\_2\_STATE **=** 6'b100001**;** // Player one Winner screen of the Two player version of the game.  **localparam** R\_2\_STATE **=** 6'b100010**;** // Player two Winner screen of the Two player version of the game.    **always** **@** **(** game\_states**,** lcd\_pixel\_y\_address**,** lcd\_pixel\_x\_address **)**  **begin**    **if** **(** PLAYER\_ID **==** 1 **)**  **begin**  **if** **(** game\_states **==** E\_1\_STATE **||** game\_states **==** G\_1\_STATE **||** game\_states **==** I\_1\_STATE **||** game\_states **==** E\_2\_STATE **||** game\_states **==** G\_2\_STATE **||** game\_states **==** I\_2\_STATE**)**  **begin**    **localparam** LCD\_WIDTH **=** 240**;**  **localparam** LCD\_HEIGHT **=** 320**;**  //1P  **localparam** GUNMAN\_X **=** 115**;**  **localparam** GUNMAN\_Y **=** 250**;**  **localparam** BODY\_COLOUR **=** 16'h0000**;**  //localparam SCARF\_COLOUR = 16'h07E0;  **localparam** BELT\_COLOUR **=** 16'hDA22**;**  **localparam** GUN\_COLOUR **=** 16'hA516**;**  **localparam** PANTS\_COLOUR **=** 16'h001F**;**  **localparam** EXPLODE **=** 16'hF800**;**  //2P  **localparam** GUNMAN2\_X **=** 115**;**  **localparam** GUNMAN2\_Y **=** 50**;**  **localparam** BODY2\_COLOUR **=** 16'h0000**;**  //localparam SCARF2\_COLOUR = 16'hF81F;  **localparam** BELT2\_COLOUR **=** 16'hDA22**;**  **localparam** GUN2\_COLOUR **=** 16'hA516**;**  **localparam** PANTS2\_COLOUR **=** 16'h001F**;**  **localparam** EXPLODE2 **=** 16'hF800**;**  //HEAD  **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**6**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**7**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**5**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**7**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**10**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**7**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**10**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**5**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**12**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**5**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN\_X**+**12**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**8**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**10**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**8**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**8**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**10**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**9**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**    //green scarf h07E0  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**6**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**11**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**6**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**11**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**8**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**9**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**19**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**8**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**9**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF\_COLOUR**;**  **end**    //brown belt  //DA22  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**23**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BELT\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**24**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BELT\_COLOUR**;**  **end**    //Gunman Body    **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**19**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**21**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**22**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**23**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**24**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**    //Arms  **else** **if** **(((**lcd\_pixel\_x\_address **<=**GUNMAN\_X**-**1**)** **&&** **(**lcd\_pixel\_x\_address **>=** **(**GUNMAN\_X**-**2**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **<=**GUNMAN\_X**-**1**)** **&&** **(**lcd\_pixel\_x\_address **>=** **(**GUNMAN\_X**-**2**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**18**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**19**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN\_X**+**18**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN\_X**+**19**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**-**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**-**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**21**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY\_COLOUR**;**  **end**        //silver guns  //A516  //left  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**0**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  //right  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN\_COLOUR**;**  **end**    //blue legs h001F  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**5**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**12**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**13**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS\_COLOUR**;**  **end**          //HEAD  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**6**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**7**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**5**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**7**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**10**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**7**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**10**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**5**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**12**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**4**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**5**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=(**GUNMAN2\_X**+**12**))** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**8**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**10**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**8**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**8**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**10**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**9**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**    //scarf  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**6**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**11**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**6**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**11**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**8**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**9**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**19**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**8**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**9**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** SCARF2\_COLOUR**;**  **end**    //brown belt  //DA22  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**23**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BELT2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**24**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BELT2\_COLOUR**;**  **end**    //GUNMAN2 Body    **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**17**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**19**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**21**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**22**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**23**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**4**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**13**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**24**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**    //Arms  **else** **if** **(((**lcd\_pixel\_x\_address **<=**GUNMAN2\_X**-**1**)** **&&** **(**lcd\_pixel\_x\_address **>=** **(**GUNMAN2\_X**-**2**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **<=**GUNMAN2\_X**-**1**)** **&&** **(**lcd\_pixel\_x\_address **>=** **(**GUNMAN2\_X**-**2**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**18**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**19**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_x\_address **>=**GUNMAN2\_X**+**18**)** **&&** **(**lcd\_pixel\_x\_address **<=** **(**GUNMAN2\_X**+**19**)))&&(**lcd\_pixel\_y\_address **==** **(**GUNMAN2\_Y**+**18**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**-**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**-**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**20**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**19**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**21**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** BODY2\_COLOUR**;**  **end**        //silver guns  //A516  //left  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**0**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  //right  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**16**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**24**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**17**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**23**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**28**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** GUN2\_COLOUR**;**  **end**    //blue legs h001F  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**4**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**5**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**12**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**25**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**13**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**    **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**2**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**3**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**1**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**14**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else** **if** **(((**lcd\_pixel\_y\_address **>=(**GUNMAN2\_Y**+**32**))** **&&** **(**lcd\_pixel\_y\_address **<=** **(**GUNMAN2\_Y**+**33**)))&&(**lcd\_pixel\_x\_address **==** **(**GUNMAN2\_X**+**15**)))**  **begin**  lcd\_pixel\_data**[**15**:**0**]** **<=** PANTS2\_COLOUR**;**  **end**  **else**  **begin**  lcd\_pixel\_data **<=** 16'hFFFF**;**  **end**      **end**  /\*End of In-Between Screens\*/    **end**    **end**    **endmodule** // End of the module. |

|  |
| --- |
| **SECTION 13:** Video Engine Test Bench |
| `timescale 1 ns**/**100 ps  // Test bench module declaration  // Always end test bench module names with \_tb for clarity  // There is no port list for a test bench  **module** LT24Top\_tb**;**  //  // Parameter Declarations  //  **localparam** NUM\_CYCLES **=** 1000000000**;** //Run simulation for NUM\_CYCLES clock cycles. (Max 1 billion)  **localparam** CLOCK\_FREQ **=** 50000000**;** //Clock frequency  **localparam** REPEAT\_DUTY\_CYCLES **=** 2000000**;**  //  // Test Bench Generated Signals  //  **reg** clock**;**  **reg** reset**;**  **reg** game\_touch\_1\_button **,** // The touch input on screen 1.  **reg** game\_touch\_2\_button **,** // The touch input on screen 2.  **reg** game\_up\_button **,** // To switch between one player and two player version.  **reg** game\_down\_button **,** // To switch between one player and two player version.  **reg** game\_select\_button **,** // To select between one player or two player version.    **wire** output\_game\_touch\_button **,** // The touch input on screen 1.  **wire** output\_game\_up\_button **,** // To switch between one player and two player version.  **wire** output\_game\_down\_button **,** // To switch between one player and two player version.  **wire** output\_game\_select\_button **,** // To select between one player or two player version.  **wire** output\_clock **,**  **wire** output\_reset **,**    // - Application Reset - for debug  // wire resetApp,    **wire** **[**6**:**0**]**seven\_segment\_0**,**  **wire** **[**6**:**0**]**seven\_segment\_1**,**  **wire** **[**6**:**0**]**seven\_segment\_2**,**  **wire** **[**6**:**0**]**seven\_segment\_3**,**  **wire** **[**6**:**0**]**seven\_segment\_4**,**  **wire** **[**6**:**0**]**seven\_segment\_5**,**  //  // DUT Output Signals  //  **wire** resetApp**;**  // LT24 Display Interface  **wire** LT24Wr\_n**;**  **wire** LT24Rd\_n**;**  **wire** LT24CS\_n**;**  **wire** LT24RS**;**  **wire** LT24Reset\_n**;**  **wire** **[**15**:**0**]** LT24Data**;**  **wire** LT24LCDOn**;**  //  // Device Under Test  //  LT24Top LT24Top\_dut **(**  **.**clock **(** clock **),**  **.**globalReset **(** reset **),**  **.**resetApp **(** resetApp **),**    **.**game\_touch\_1\_button **(** game\_touch\_1\_button **),**  **.**game\_touch\_2\_button **(**game\_touch\_2\_button**),**  **.**game\_up\_button **(**game\_up\_button**),**  **.**game\_down\_button **(**game\_down\_button**),**  **.**game\_select\_button **(**game\_select\_button**),**  **.**output\_game\_touch\_button **(**output\_game\_touch\_button**),**  **.**output\_game\_up\_button **(**output\_game\_up\_button**),**  **.**output\_game\_down\_button **(**output\_game\_down\_button**),**  **.**output\_game\_select\_button **(**output\_game\_select\_button**),**  **.**output\_clock **(**output\_clock**),**  **.**output\_reset **(**output\_reset**),**  **.**seven\_segment\_0 **(**seven\_segment\_0**),**  **.**seven\_segment\_1 **(**seven\_segment\_1**),**  **.**seven\_segment\_2 **(**seven\_segment\_2**),**  **.**seven\_segment\_3 **(**seven\_segment\_3**),**  **.**seven\_segment\_4 **(**seven\_segment\_4**),**  **.**seven\_segment\_5 **(**seven\_segment\_5**),**    **.**LT24Wr\_n **(** LT24Wr\_n **),**  **.**LT24Rd\_n **(** LT24Rd\_n **),**  **.**LT24CS\_n **(** LT24CS\_n **),**  **.**LT24RS **(** LT24RS **),**  **.**LT24Reset\_n **(** LT24Reset\_n **),**  **.**LT24Data **(** LT24Data **),**  **.**LT24LCDOn **(** LT24LCDOn **),**    //.lcd\_pixel\_x\_address (lcd\_pixel\_x\_address ) , // The current x-coordinate of the LCD pixel.  //.lcd\_pixel\_y\_address ( , // The current y-coordinate of the LCD pixel.  //.game\_states , // The current state of the game screen.  //.lcd\_pixel\_data  **);**  //  // Display Functional Model  //  LT24FunctionalModel **#(**  **.**WIDTH **(** 240 **),**  **.**HEIGHT **(** 320 **)**  **)** DisplayModel **(**  // LT24 Interface  **.**LT24Wr\_n **(** LT24Wr\_n **),**  **.**LT24Rd\_n **(** LT24Rd\_n **),**  **.**LT24CS\_n **(** LT24CS\_n **),**  **.**LT24RS **(** LT24RS **),**  **.**LT24Reset\_n **(** LT24Reset\_n **),**  **.**LT24Data **(** LT24Data **),**  **.**LT24LCDOn **(** LT24LCDOn **)**  **);**  //  // Test Bench Logic  //  **initial** **begin**  $display**(**"%d ns\tSimulation Started"**,**$time**);** //Print to console that the simulation has started  reset **=** 1'b1**;** //Start in reset.  **repeat(**2**)** **@(posedge** clock**);** //Wait for a couple of clock cycles  reset **=** 1'b0**;** //Then clear the reset signal.  **wait(**resetApp **===** 1'b0**);** //Wait until the resetApp signal is zero.  $display**(**"%d ns\tInitialisation Complete"**,**$time**);** //Print to console that initialisation of the display is complete.      **end**  //  //Clock generator + simulation time limit.  //  **initial** **begin**  clock **=** 1'b0**;** //Initialise the clock to zero.  **end**  //Next we convert our clock period to nanoseconds and half it  //to work out how long we must delay for each half clock cycle  **real** HALF\_CLOCK\_PERIOD **=** **(**1000000000.0 **/** $itor**(**CLOCK\_FREQ**))** **/** 2.0**;**  //Now generate the clock  **integer** half\_cycles **=** 0**;**  **always** **begin**  //Generate the next half cycle of clock  **#(**HALF\_CLOCK\_PERIOD**);** //Delay for half a clock period.  clock **=** **~**clock**;** //Toggle the clock  half\_cycles **=** half\_cycles **+** 1**;** //Increment the counter  /\*    repeat ( REPEAT\_DUTY\_CYCLES ) // Repeat the loop for some time.  begin    TB\_DUTY\_CYCLE\_CONTROL = 8'd0; // Assign a value of 0 for the duty cycle control.    /\* Generating individual half cycles of clock \*/  /\*#(HALF\_CLOCK\_PERIOD); // Delay for half a clock period.  TB\_CLOCK = ~ TB\_CLOCK; // Toggle the clock signal.  TB\_SERVO\_SELECTOR = 2'd0; // Select the first servo.  TB\_LOAD\_SIGNAL = 1'b1; // Set the load signal to HIGH so that the output latches on to the required duty cycle.  half\_cycles = half\_cycles + 1; // Increment the count of number of half cycles.    end  game\_touch\_1\_button = 1'b0;  #2;  game\_touch\_1\_button = 1'b1;  #2;  game\_touch\_1\_button = 1'b0;  #2;  repeat ( REPEAT\_DUTY\_CYCLES ) // Repeat the loop for some time.  begin    TB\_DUTY\_CYCLE\_CONTROL = 8'd0; // Assign a value of 0 for the duty cycle control.    /\* Generating individual half cycles of clock \*/  /\*#(HALF\_CLOCK\_PERIOD); // Delay for half a clock period.  TB\_CLOCK = ~ TB\_CLOCK; // Toggle the clock signal.  TB\_SERVO\_SELECTOR = 2'd0; // Select the first servo.  TB\_LOAD\_SIGNAL = 1'b1; // Set the load signal to HIGH so that the output latches on to the required duty cycle.  half\_cycles = half\_cycles + 1; // Increment the count of number of half cycles.    end  game\_select\_button = 1'b0;  #2;  game\_select\_button = 1'b1;  #2;  game\_select\_button = 1'b0;  #2;  \*/    //Check if enough half clock cycle  **if** **(**half\_cycles **==** **(**2**\***NUM\_CYCLES**))** **begin**  //Once the number of cycles has been reached  half\_cycles **=** 0**;** //Reset half cycles, so if we resume running with "run -all", we perform another chunk.  $stop**;** //Break the simulation  //Note: We can continue the simulation after this breakpoint using "run -continue" or "run x ns" or "run -all" in modelsim.  **end**  **end**  **endmodule** |

|  |
| --- |
| **SECTION 14:** Ready Steady Bang |
| ////////////////////////////////////////////////////////////////////////////////  /\*  FPGA Project Name : Ready Steady Bang  Top level Entity Name : Ready\_Steady\_Bang  Target Device : Cyclone V    Code Authors : Sanjith Chandran and Shrajan Bhandary  Date Created : 20/04/2019  Location : University of Leeds  Module : ELEC5566M FPGA Design for System-on-chip    -------------------------------------------------------------------------------    Description of the Verilog Module:  The module is the main control panel of the game. All the control inputs and  screen states are manipulated to create game-play. The inputs are fed to the  game engine that determines the state and passes this information to the  video engine. Along with state values, it also passes player scores to the  seven segment display controllers. The video engine accepts the state values  from the game engine and generates corresponding pixel data depending on the  co-ordinates. This pixel data is fed to the LT24 display module that then  showcases different screens and makes up for the complete gameplay.    \*/  ////////////////////////////////////////////////////////////////////////////////  /\* ceil(log2(N)) Preprocessor Macro \*/  `define clog2**(**x**)** **(** \  **((**x**)** **<=** 2**)** **?** 1 **:** \  **((**x**)** **<=** 4**)** **?** 2 **:** \  **((**x**)** **<=** 8**)** **?** 3 **:** \  **((**x**)** **<=** 16**)** **?** 4 **:** \  **((**x**)** **<=** 32**)** **?** 5 **:** \  **((**x**)** **<=** 64**)** **?** 6 **:** \  **((**x**)** **<=** 128**)** **?** 7 **:** \  **((**x**)** **<=** 256**)** **?** 8 **:** \  **((**x**)** **<=** 512**)** **?** 9 **:** \  **((**x**)** **<=** 1024**)** **?** 10 **:** \  **((**x**)** **<=** 2048**)** **?** 11 **:** \  **((**x**)** **<=** 4096**)** **?** 12 **:** 16**)**  **module** Ready\_Steady\_Bang **#** **(**    /\* Parameter List of the Ready\_Steady\_Bang \*/  **parameter** PLAYER\_ID **=** 1 **,** // The ID is either 1 or 2 depending on the board that the program runs.    /\* Parameters required for video engine. \*/  **parameter** LCD\_WIDTH **=** 240 **,** // The number of pixels in the x-direction is 240.  **parameter** LCD\_HEIGHT **=** 320 **,** // The number of pixels in the y-direction is 320.  **parameter** X\_ADDRESS\_WIDTH **=** `clog2**(**LCD\_WIDTH**)** **,** // Number of bits required to store the x-address.  **parameter** Y\_ADDRESS\_WIDTH **=** `clog2**(**LCD\_HEIGHT**),** // Number of bits required to store the y-address.  **parameter** PIXEL\_ADDRESS\_WIDTH **=** 16 **,** // Number of bits required to colour a pixel.    /\* Parameters required for game engine and seven segment display. \*/  **parameter** INVERTED\_INPUT **=** 1 **,** // Parameter to select between active LOW ( Invert = 1 ) inputs and active HIGH ( Invert = 0 )inputs.  **parameter** INVERTED\_OUTPUT **=** 1 **,** // Parameter to select between active LOW ( Invert = 1 ) outputs and active HIGH ( Invert = 0 )outputs.  **parameter** INCOMING\_CLOCK\_FREQUENCY **=** 50000000**,** // The frequency of the board is 50 MHz.  **parameter** REDUCED\_CLOCK\_FREQUENCY **=** 10 **,** // The fixed operable frequency is set to 2 Hz (Every clock cycle is 0.1 s long).    **parameter** NO\_GAME\_STATES **=** 35 **,** // The number of possible states (Screens).  **parameter** STATE\_ADDRESS\_WIDTH **=** `clog2**(**NO\_GAME\_STATES**),** // Number of bits required to store the different number of states.    **parameter** MAXIMUM\_BANG\_TRIGGER **=** 10 **,** // The maximum limit of time in seconds when BANG is triggered.  **parameter** MINIMUM\_BANG\_TRIGGER **=** 2 **,** // The minimum limit of time in seconds when BANG is triggered.  **parameter** WINNING\_SCORE **=** 5 **,** // The game ends when either one of the player is the first to score the winning points.  **parameter** STATE\_TIME\_GAP **=** 1 // The time gap in seconds between start screen, ready screen and steady screen.  **)(**  /\* Port List of the Game\_Engine \*/    /\* Clock and Resets Inputs. \*/  **input** clock **,** // The incoming clock of the FPGA board is connected to this port as master game clock.  **input** globalReset **,** // The master reset pin of the game is connected to this port.  **output** resetApp **,** // This reset is used for debugging.    /\* Game-play control inputs. \*/  **input** game\_touch\_1\_button **,** // The touch input on screen 1.  **input** game\_touch\_2\_button **,** // The touch input on screen 2.  **input** game\_up\_button **,** // To switch between one player and two player version.  **input** game\_down\_button **,** // To switch between one player and two player version.  **input** game\_select\_button **,** // To select between one player or two player version.    /\* Pins to control the second board using the first board. \*/  **output** **reg** output\_game\_touch\_button **,** // The touch input of the screen.  **output** output\_game\_up\_button **,** // To switch between one player and two player version.  **output** output\_game\_down\_button **,** // To switch between one player and two player version.  **output** output\_game\_select\_button **,** // To select between one player or two player version.  **output** output\_clock **,** // The master clock is supplied to the secondary board.  **output** output\_reset **,** // The master reset is supplied to the secondary board.    /\* The pins to control the seven segment screens and display the scores of the players. \*/  **output** **[**6**:**0**]** seven\_segment\_0 **,** // The score of the first player is displayed on this segment.  **output** **[**6**:**0**]** seven\_segment\_1 **,** // The letter 'b' is displayed on this segment.  **output** **[**6**:**0**]** seven\_segment\_2 **,** // The letter 'P' is displayed on this segment.  **output** **[**6**:**0**]** seven\_segment\_3 **,** // The score of the second player is displayed on this segment.  **output** **[**6**:**0**]** seven\_segment\_4 **,** // The letter 'A' is displayed on this segment.  **output** **[**6**:**0**]** seven\_segment\_5 **,** // The letter 'P' is displayed on this segment.    /\* LT24 Interface signals. \*/  **output** LT24Wr\_n **,**  **output** LT24Rd\_n **,**  **output** LT24CS\_n **,**  **output** LT24RS **,**  **output** LT24Reset\_n **,**  **output** **[**15**:**0**]** LT24Data **,**  **output** LT24LCDOn  **);**  /\* For two player version, two boards are connected to each other via GPIO's. \*/  **assign** output\_game\_up\_button **=** game\_up\_button**;**  **assign** output\_game\_down\_button **=** game\_down\_button**;**  **assign** output\_game\_select\_button **=** game\_select\_button**;**  **assign** output\_clock **=** clock**;**  **assign** output\_reset **=** globalReset**;**    /\* Local Variables to control the LT24 LCD display. \*/  **wire** **[** 7**:**0**]** xAddr**;**  **wire** **[** 8**:**0**]** yAddr**;**  **reg** **[**15**:**0**]** pixelData**;**  **wire** pixelReady**;**  **reg** pixelWrite**;**  **wire** **[(**PIXEL\_ADDRESS\_WIDTH**-**1**):**0**]** state\_pixel\_value**;** // This net is connected to the video engine module to obtain pixel data according to game state and co-ordinates.    **wire** **[(**STATE\_ADDRESS\_WIDTH**-**1**):**0**]** state\_value**;** // This net obtains current state of the game and is used to generate the game screens.    /\* The nets that determine what is display on the seven segment display. \*/  **wire** **[**3**:**0**]** seven\_segment\_value\_0**;** // This net is connected to the game engine module to obtain the score of player two.  **wire** **[**3**:**0**]** seven\_segment\_value\_1 **=** 4'b1011**;** // The value for letter 'b'.  **wire** **[**3**:**0**]** seven\_segment\_value\_2 **=** 4'b1111**;** // The value for letter 'P'.  **wire** **[**3**:**0**]** seven\_segment\_value\_3**;** // This net is connected to the game engine module to obtain the score of player one.  **wire** **[**3**:**0**]** seven\_segment\_value\_4 **=** 4'b1010**;** // The value for letter 'A'.  **wire** **[**3**:**0**]** seven\_segment\_value\_5 **=** 4'b1111**;** // The value for letter 'P'.    /\* Controlling the LT24 LCD display. \*/  LT24Display **#(**  **.**WIDTH **(**LCD\_WIDTH **),**  **.**HEIGHT **(**LCD\_HEIGHT **),**  **.**CLOCK\_FREQ **(**50000000 **)**  **)** Display **(**  **.**clock **(**clock **),**  **.**globalReset **(**globalReset**),**  **.**resetApp **(**resetApp **),**  **.**xAddr **(**xAddr **),**  **.**yAddr **(**yAddr **),**  **.**pixelData **(**pixelData **),**  **.**pixelWrite **(**pixelWrite **),**  **.**pixelReady **(**pixelReady **),**  **.**pixelRawMode**(**1'b0 **),**  **.**cmdData **(**8'b0 **),**  **.**cmdWrite **(**1'b0 **),**  **.**cmdDone **(**1'b0 **),**  **.**cmdReady **(** **),**  **.**LT24Wr\_n **(**LT24Wr\_n **),**  **.**LT24Rd\_n **(**LT24Rd\_n **),**  **.**LT24CS\_n **(**LT24CS\_n **),**  **.**LT24RS **(**LT24RS **),**  **.**LT24Reset\_n **(**LT24Reset\_n**),**  **.**LT24Data **(**LT24Data **),**  **.**LT24LCDOn **(**LT24LCDOn **)**  **);**  /\* The x-coordinate of the LCD increases with every rise of clock. \*/  UpCounterNbit **#(**  **.**WIDTH **(** 8**),**  **.**MAX\_VALUE**(**LCD\_WIDTH**-**1**)**  **)** xCounter **(**  **.**clock **(**clock **),**  **.**reset **(**resetApp **),**  **.**enable **(**pixelReady**),**  **.**countValue**(**xAddr **)**  **);**  /\* The y-coordinate of the LCD increases with every rise of clock. \*/  **wire** yCntEnable **=** pixelReady **&&** **(**xAddr **==** **(**LCD\_WIDTH**-**1**));**  UpCounterNbit **#(**  **.**WIDTH **(** 9**),**  **.**MAX\_VALUE**(**LCD\_HEIGHT**-**1**)**  **)** yCounter **(**  **.**clock **(**clock **),**  **.**reset **(**resetApp **),**  **.**enable **(**yCntEnable**),**  **.**countValue**(**yAddr **)**  **);**  /\* The 16-bit data that has to be written to the LCD display. \*/  **always** **@** **(posedge** clock **or** **posedge** resetApp**)**  **begin**  **if** **(**resetApp**)**  **begin**  pixelWrite **<=** 1'b0**;**  **end**  **else**  **begin**  pixelWrite **<=** 1'b1**;**  **end**  **end**  /\* Creating different colour for each pixel based on the X-Y coordinate. \*/  **always** **@** **(posedge** clock **or** **posedge** resetApp**)** **begin**  **if** **(**resetApp**)**  **begin**  pixelData **<=** 16'b0**;**  **end**  **else** **if** **(**pixelReady**)**  **begin**  pixelData **<=** state\_pixel\_value**;**  **end**  **end**    /\* For two player version, two boards are connected to each other via GPIO's. \*/  **always** **@** **(** **posedge** clock **)**  **begin**    **if** **(** PLAYER\_ID **==** 1 **)**  **begin**  output\_game\_touch\_button **<=** game\_touch\_1\_button**;**  **end**    **else** **if** **(** PLAYER\_ID **==** 2 **)**  **begin**  output\_game\_touch\_button **<=** game\_touch\_2\_button**;**  **end**  **end**    /\* Instantiating the game engine to control the game and generate states. \*/  Game\_Engine **#** **(**    **.**INVERTED\_INPUT **(** INVERTED\_INPUT **),**  **.**INVERTED\_OUTPUT **(** INVERTED\_OUTPUT **),**  **.**INCOMING\_CLOCK\_FREQUENCY **(** INCOMING\_CLOCK\_FREQUENCY **),**  **.**REDUCED\_CLOCK\_FREQUENCY **(** REDUCED\_CLOCK\_FREQUENCY **),**  **.**NO\_GAME\_STATES **(** NO\_GAME\_STATES **),**  **.**MAXIMUM\_BANG\_TRIGGER **(** MAXIMUM\_BANG\_TRIGGER **),**  **.**MINIMUM\_BANG\_TRIGGER **(** MINIMUM\_BANG\_TRIGGER **),**  **.**WINNING\_SCORE **(** WINNING\_SCORE **),**  **.**STATE\_TIME\_GAP **(** STATE\_TIME\_GAP **)**    **)** GAME\_GENERATOR **(**    **.**game\_clock\_in **(** clock **),**  **.**game\_reset\_in **(** **~**globalReset **),**  **.**game\_touch\_1\_button **(** game\_touch\_1\_button **),**  **.**game\_touch\_2\_button **(** game\_touch\_2\_button **),**  **.**game\_up\_button **(** game\_up\_button **),**  **.**game\_down\_button **(** game\_down\_button **),**  **.**game\_select\_button **(** game\_select\_button **),**  **.**display\_game\_state **(** state\_value **),**  **.**p\_1\_score **(** seven\_segment\_value\_3 **),**  **.**p\_2\_score **(** seven\_segment\_value\_0 **)**  **);**    /\* Instantiating the video engine to control the screen by generating pixel data. \*/  Video\_Engine **#** **(**    **.**PLAYER\_ID **(** PLAYER\_ID **),**  **.**LCD\_WIDTH **(** LCD\_WIDTH **),**  **.**LCD\_HEIGHT **(** LCD\_HEIGHT **),**  **.**X\_ADDRESS\_WIDTH **(** X\_ADDRESS\_WIDTH **),**  **.**Y\_ADDRESS\_WIDTH **(** Y\_ADDRESS\_WIDTH **),**  **.**PIXEL\_ADDRESS\_WIDTH **(** PIXEL\_ADDRESS\_WIDTH **)**    **)** VIDEO\_GENERATOR **(**    **.**lcd\_pixel\_x\_address **(** xAddr **),**  **.**lcd\_pixel\_y\_address **(** yAddr **),**  **.**game\_states **(** state\_value **),**  **.**lcd\_pixel\_data **(** state\_pixel\_value **)**  **);**    /\* The scores of the players are displayed using seven segment controller. \*/    /\* The value for second player score. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_0  **(**  **.**num\_hex **(** seven\_segment\_value\_0 **),**  **.**ss\_hex **(** seven\_segment\_0 **)**  **);**    /\* The value for letter 'b'. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_1  **(**  **.**num\_hex **(** seven\_segment\_value\_1 **),**  **.**ss\_hex **(** seven\_segment\_1 **)**  **);**    /\* The value for letter 'P'. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_2  **(**  **.**num\_hex **(** seven\_segment\_value\_2 **),**  **.**ss\_hex **(** seven\_segment\_2 **)**  **);**    /\* The value for first player score. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_3  **(**  **.**num\_hex **(** seven\_segment\_value\_3 **),**  **.**ss\_hex **(** seven\_segment\_3 **)**  **);**    /\* The value for letter 'A'. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_4  **(**  **.**num\_hex **(** seven\_segment\_value\_4 **),**  **.**ss\_hex **(** seven\_segment\_4 **)**  **);**    /\* The value for letter 'P'. \*/  Seven\_Segment\_Controller **#**  **(**  **.**INVERT\_OUTPUT **(** INVERTED\_OUTPUT **)**  **)**  SEVEN\_SEGMENT\_DISPLAY\_5  **(**  **.**num\_hex **(** seven\_segment\_value\_5 **),**  **.**ss\_hex **(** seven\_segment\_5 **)**  **);**    **endmodule** |

**APPENDIX – B**

**Examples of Verilog test benches that were used for testing the project.**

|  |
| --- |
| **SECTION 1:** Drawing words in a white colour i.e. READY  ../Desktop/Local%20Parameter%20for%20Ready%20Word.png../Desktop/Ready-word%20Location.png../Desktop/White%20Pixel%20for%20Ready%20Word%20for%20the%20Start%20Scre |

|  |
| --- |
| **SECTION 2:** Filling in the remaining pixels with a background colour where there is no word or gunman character being displayed  ../Desktop/Background%20Colour%20of%20Black%20for%20Start%20Screen.p../Desktop/Black%20Pixels%20for%20background%20i.e.%20no%20character%20image%20or%20 |

**APPENDIX – C**

**Hardware verification images of the entire gameplay.**

|  |
| --- |
| **SECTION 1:** Main screen of the game. |
| FPGA%20MINI%20PROJECT%20IMAGES/Ready,%20Steady,%20Bang.jpg |

|  |
| --- |
| **SECTION 2:** One player and two player mode screens of the game. |
| FPGA%20MINI%20PROJECT%20IMAGES/Select%20Mode%20-%201P%20.jpgFPGA%20MINI%20PROJECT%20IMAGES/Select%20Mode%20-%202P.jpg |

|  |
| --- |
| **SECTION 3:** Start (left) and empty (right) screens of the game. |
| FPGA%20MINI%20PROJECT%20IMAGES/START.jpgFPGA%20MINI%20PROJECT%20IMAGES/DELAY%20SCREEN.jpg |

|  |
| --- |
| **SECTION 4:** Ready, Steady and Bang Screens. |
| FPGA%20MINI%20PROJECT%20IMAGES/READY.jpgFPGA%20MINI%20PROJECT%20IMAGES/STEADY.jpgFPGA%20MINI%20PROJECT%20IMAGES/BANG.jpg |

|  |
| --- |
| **SECTION 5:** Next and Winner Screens. |
| FPGA%20MINI%20PROJECT%20IMAGES/NEXT.jpgFPGA%20MINI%20PROJECT%20IMAGES/WINNER%20.jpg |

|  |
| --- |
| **SECTION 6:** Scores using seven segment display. |
| FPGA%20MINI%20PROJECT%20IMAGES/Score%20Display.jpg |

|  |
| --- |
| **SECTION 7:** Hardware implementation of one player mode.  **FPGA%20MINI%20PROJECT%20IMAGES/MIDGAME%20SCORE.jpg** |
| FPGA%20MINI%20PROJECT%20IMAGES/WINNER%20WITH%20SCORE.jpg |

|  |
| --- |
| **SECTION 8:** Control inputs of the game. |
| FPGA%20MINI%20PROJECT%20IMAGES/HARDWARE%20INPUT.jpg |

|  |
| --- |
| **SECTION 9:** GPIO 1 pin configuration. |
| FPGA%20MINI%20PROJECT%20IMAGES/GPIO-1%20SCHEMATIC.png |

|  |
| --- |
| **SECTION 10:** Hardware implementation of two player mode. |
| FPGA%20MINI%20PROJECT%20IMAGES/MULTI%20-%20BANG.jpg  FPGA%20MINI%20PROJECT%20IMAGES/MULTIPLAYER%20MODE.jpg |

|  |
| --- |
| **SECTION 11:** Hardware implementation of two player mode. |
| FPGA%20MINI%20PROJECT%20IMAGES/SHRAJAN%20&%20SANJITH%20BOARDS%20-%20CONNECTING%202%20DE1-SOC%20BOARDS.jpg |