Hot Wheels Game Report

Shadman Kaif & Abdurrafay Khan
2019 Fall Instance of ECE241
Teaching Assistant: Mohamed Elgammal

Table of Contents

1.0 Introduction

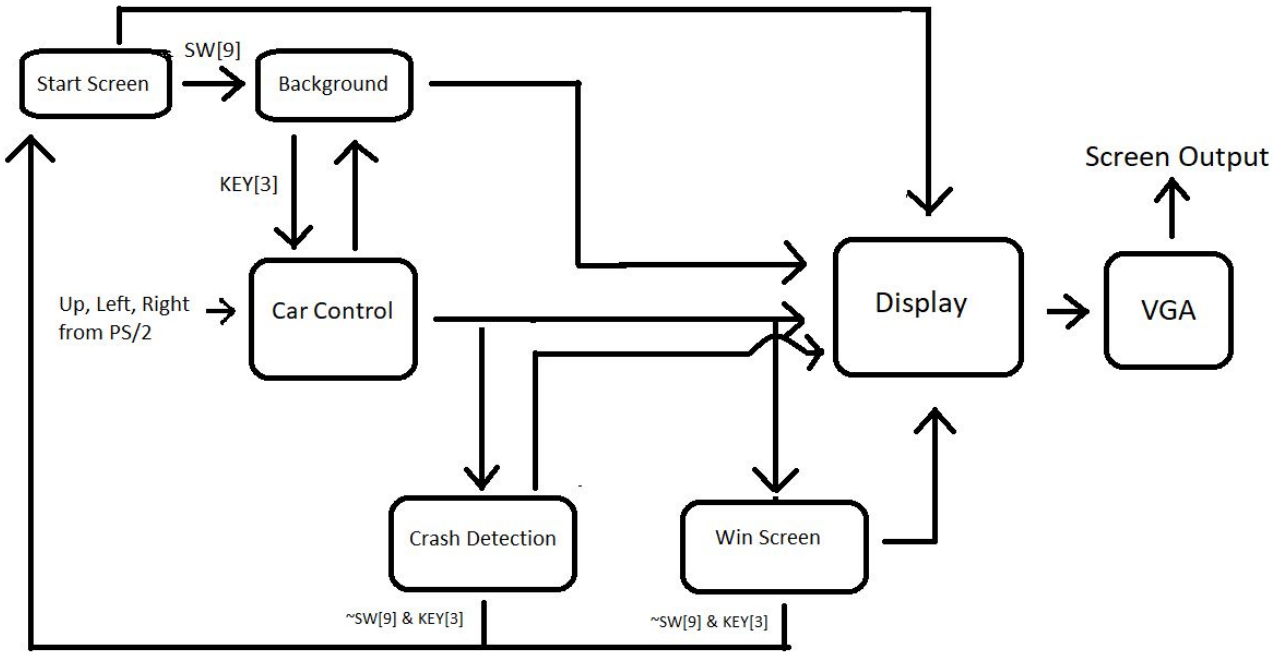
Growing up, both of us were avid fans of the popular Gameloft car racing game, “Asphalt Legends”. Thus, when faced with the task of choosing a project for our Digital Systems class, we knew that a car racing game would be both fun and challenging to create. Initially, we wanted to create a bird’s eye view game of “Crazy Taxi”, featuring a user-controlled taxi that would evade obstacles in the form of other cars that would come on the road in waves, varying in both speed and location. However, this idea did not entirely come into fruition as we realized that it would be difficult to incorporate several obstacles coming in from outside of the background at different speeds and times in the game.

Eventually, we settled on our new idea: a car racing game where a user-controlled car needs to finish a lap around a race track. If the car comes in contact with the grass that is not apart of the race track, there is an explosion and the game is over. We generated five Read Only Memory (ROM) files using Quartus’ IP Catalog in order to make the game more realistic. The five images consisted of: the car, the race track, the explosion, the start screen and the win screen. Moreover, we programmed the car such that there were three directions that it could move - forward, left and right. These three inputs were read using a finite state machine (FSM) from the PS/2 keyboard that the user would press. Depending on the user input, the car would rotate 22.5 degrees clockwise or counterclockwise varying based on the 16 different orientations (from 0 degrees to 337.5 degrees). These degree rotations were generated through manipulation of the car’s “x” and “y” position movements.

Our goal was to create a single player car racing game that is both challenging and enjoyable. We wanted to incorporate a certain score pertaining to each victorious run of the game, conveyed through our counter on the hex display. We wanted to maximize our game’s features by incorporating a plethora of topics covered in ECE241 this semester.

2.0 The Design

2.1.1 **Gamestate Block Diagram**



2.1.2 **Description of Gamestate Block Diagram**

The start screen is read from ROM and displayed on the screen. In order to proceed, the user flips SW[9] and resets using KEY[3]. The background is then displayed, with the car at the start line. Using the up, left and right arrow keys on the PS/2 keyboard, which was utilized through the means of a FSM, the car can move forward, left and right. This is the bulk of the game, where the car can rotate 22.5 degrees and move, depending on the user input.

If, however, the car comes into contact with the background colour of green (6'b001001) that is not apart of the road, an explosion will be displayed on the screen, thus replacing the car. In this case, the game has come to an end and the user is unable to move the car using the arrow keys. The user can reset the game and attain the start screen by flipping off SW[9] and pressing KEY[3] to reset.

If the user is able to maneuver the car through the racetrack and towards the finish lane, they will have finished the game and the win screen is displayed. The user will have to flip off SW[9] and reset using KEY[3] in order to display the start screen again and commence a new game, similar to the aforementioned case of the explosion. As shown by our block diagram, the car control is at the centre of the gameplay, dictating whether the explosion or win screen is displayed.

To provide a debrief about each module, the win screen, start screen, car, explosion, and background (that being the racetrack) are all images that were either generated through Microsoft Paint or the Internet. These were converted to MIF files, which were then used through the IP Catalog in Quartus to create ROM files. The x and y plots were given as counters and passed through the VGA module that UofT provided us in Lab 7 Part 2, thus being able to display each pixel of every ROM image that we intended to display. The crash was detected through the car touching the green colour of the background and the win was detected when the car passed both a certain marker in the race track and the finish line.

2.2.1 **Counter Block Diagram**

2.2.2 **Description of Counter Block Diagram**

We have a seconds countdown module that counts down from a very large number to 0. Then we have a wire assignment, called Enable One Second, that checks the status of the output of the seconds countdown. If seconds countdown is 0, Enable One Second is set to true and at every posedge of the clock, it adds one second if the race is not finished. If the race if finished, the seconds passed holds its previous value. The seconds are taken into account by creating an

output register that is later instantiated with the 7 seg hex display so that we can see the run time on Hex0 and Hex1.

3.0 Report on Success

The following are images from our final project:

Figure 1: Start Screen

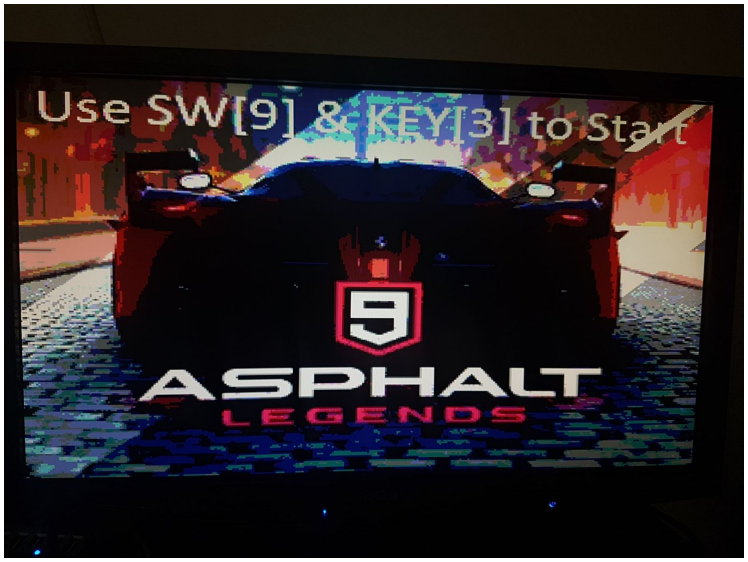


Figure 2: Car at Start Line

Figure 3: Explosion

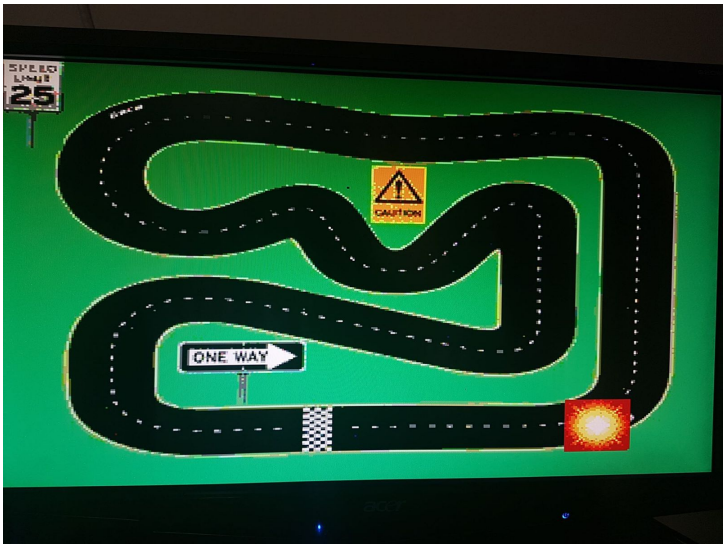


Figure 4: Win Screen

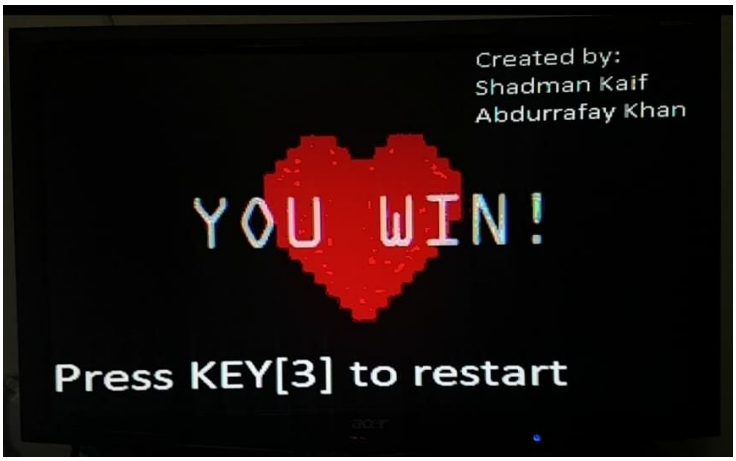
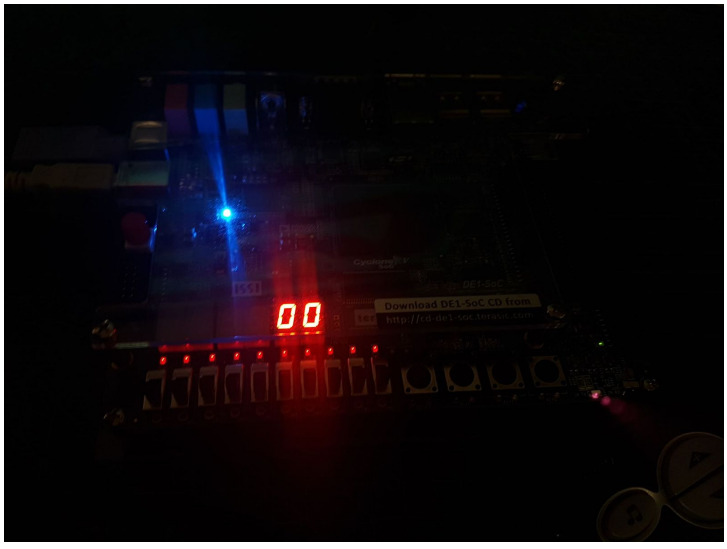
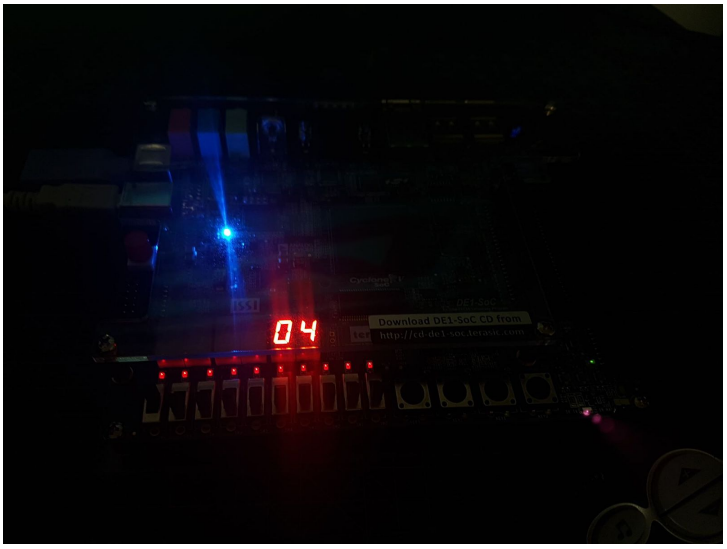


Figure 6: Counter Reading 0 Seconds vs Counter Reading a Real Time Game





Overall, we faced a myriad of problems throughout the course of this project. Most of the problems were resolved through the means of continuous debugging as we owned a DE1-SOC board at home. These problems include: the green colour not being detected for the explosion due to an inconsistent level of colouring, and the car passing the finish line but the win screen not being detected due to the car being at an even or odd pixel number which was not accounted for in our finish line. These issues were resolved by recolouring the background on Paint using a consistent green especially on the borders and detecting the finish line using both an even and odd pixel number using the "||" operator.

However, one issue that we were not able to resolve was when the car passes the start line, moves forward for a bit and the user manually rotates the car 180 degrees and moves forward towards the start line again. Once it passes the start line, the win screen is displayed as both the starting marker and the finish line marker were detected. Ultimately, we were unable to detect whether the car fairly completes an entire lap when we presented the final project to our TA. This resulted in the game having a major bug that could be easily exploited by users who are attentive. Although, we did account for this and placed a "One Way" sign at the top of the start line to convey to the player that they should not turn back, but in reality we are not able to prevent them from rotating 180 degrees and finishing the race instantaneously.

4.0 Different Approaches

Given the task of restarting this project, we would have several changes to make. Firstly, our idea of "Crazy Taxi" had a major dilemma - we did not know how to randomly incorporate obstacles coming in at different speeds at different times. However, we were able to generate a block moving right and left with an object coming down autonomously at constant speed. Thus, the concept of "Crazy Taxi" was ostensibly established. In reality, the background that we created had reverted to a constant green colour background. After communication with our TA, we realized that we needed to redraw the background for every refresh and overwriting the background parameter will just do redraw it initially. We then changed our game to one that inherently had one object moving. In a perfect world, if we had more time, we believe that we could have completed the desired "Crazy Taxi" game as we had the two essential elements covered (the car and one obstacle).

Another change that we would make corresponds to our only bug in the game. After presenting the project to our TA, we were able to resolve the issue. We thought about setting markers throughout several parts of the race track to ensure that the win screen only displays if the car passes each marker. Before, we only had one marker which was a few pixels after the start line, making it possible for the player to rotate the car and head towards the start line again. The new process of checking the markers throughout the race track would mitigate this bug and we definitely would add this feature if we were to start this project over again.

Appendix A: Car ROM

```
// megafunction
wizard: %ROM:
1-PORT%
                    // GENERATION: STANDARD
                    // VERSION: WM1.0
                    // MODULE: altsyncram


                    // ============================================================
                    // File Name: car_rom.v
                    // Megafunction Name(s):
                    //                   altsyncram
                    //
                    // Simulation Library Files(s):
                    //                   altera_mf
                    // ============================================================
                    // ************************************************************
                    // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
                    //
                    // 18.0.0 Build 614 04/24/2018 SJ Lite Edition
                    // ************************************************************



                    //Copyright (C) 2018  Intel Corporation. All rights reserved.
                    //Your use of Intel Corporation's design tools, logic functions
                    //and other software and tools, and its AMPP partner logic
                    //functions, and any output files from any of the foregoing
                    //(including device programming or simulation files), and any
                    //associated documentation or information are expressly subject
```

```
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module car_rom (
	address,
	clock,
	q);

	input	[13:0]  address;
	input	  clock;
	output	[5:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	  clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

	wire [5:0] sub_wire0;
```

```
wire [5:0] q = sub_wire0[5:0];


altsyncram     altsyncram_component (
                        .address_a (address),
                        .clock0 (clock),
                        .q_a (sub_wire0),
                        .aclr0 (1'b0),
                        .aclr1 (1'b0),
                        .address_b (1'b1),
                        .addressstall_a (1'b0),
                        .addressstall_b (1'b0),
                        .byteena_a (1'b1),
                        .byteena_b (1'b1),
                        .clock1 (1'b1),
                        .clocken0 (1'b1),
                        .clocken1 (1'b1),
                        .clocken2 (1'b1),
                        .clocken3 (1'b1),
                        .data_a ({6{1'b1}}),
                        .data_b (1'b1),
                        .eccstatus (),
                        .q_b (),
                        .rden_a (1'b1),
                        .rden_b (1'b1),
                        .wren_a (1'b0),
                        .wren_b (1'b0));
defparam
        altsyncram_component.address_aclr_a = "NONE",
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
```

```
                altsyncram_component.init_file = "car.mif",
                altsyncram_component.intended_device_family = "Cyclone
V",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 16384,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "UNREGISTERED",
                altsyncram_component.widthad_a = 14,
                altsyncram_component.width_a = 6,
                altsyncram_component.width_byteena_a = 1;



endmodule


// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
```

```
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../Desktop/Final
241/FinalVerilogProject-master/FinalVerilogProject-master/car.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "16384"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "14"
// Retrieval info: PRIVATE: WidthData NUMERIC "6"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../Desktop/Final
241/FinalVerilogProject-master/FinalVerilogProject-master/car.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone
V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "16384"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
```

```
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "14"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "6"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 14 0 INPUT NODEFVAL
"address[13..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: q 0 0 6 0 OUTPUT NODEFVAL "q[5..0]"
// Retrieval info: CONNECT: @address_a 0 0 14 0 address 0 0 14 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 6 0 @q_a 0 0 6 0
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL car_rom_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```

Appendix B: Explosion ROM

```
// megafunction wizard:
%ROM: 1-PORT%
                                    // GENERATION: STANDARD
                                    // VERSION: WM1.0
                                    // MODULE: altsyncram


                                    //
                                    ============================================================
                                    =
                                    // File Name: boom_rom.v
                                    // Megafunction Name(s):
                                    //                    altsyncram
                                    //
                                    // Simulation Library Files(s):
                                    //                    altera_mf
                                    //
                                    ============================================================
                                    =
                                    //
                                    ************************************************************
                                    *
                                    // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
                                    //
                                    // 18.0.0 Build 614 04/24/2018 SJ Lite Edition
                                    //
                                    ************************************************************
                                    *




                                    //Copyright (C) 2018  Intel Corporation. All rights
                                    reserved.
```

```
//Your use of Intel Corporation's design tools, logic
functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and
any
//associated documentation or information are expressly
subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License
Agreement,
//the Intel FPGA IP License Agreement, or other applicable
license
//agreement, including, without limitation, that your use
is for
//the sole purpose of programming logic devices
manufactured by
//Intel and sold by Intel or its authorized distributors.
Please
//refer to the applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module boom_rom (
        address,
        clock,
        q);

        input  [9:0]  address;
        input    clock;
        output [5:0]  q;
```

```
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1	clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif


	wire [5:0] sub_wire0;
	wire [5:0] q = sub_wire0[5:0];


	altsyncram	altsyncram_component (
				.address_a (address),
				.clock0 (clock),
				.q_a (sub_wire0),
				.aclr0 (1'b0),
				.aclr1 (1'b0),
				.address_b (1'b1),
				.addressstall_a (1'b0),
				.addressstall_b (1'b0),
				.byteena_a (1'b1),
				.byteena_b (1'b1),
				.clock1 (1'b1),
				.clocken0 (1'b1),
				.clocken1 (1'b1),
				.clocken2 (1'b1),
				.clocken3 (1'b1),
				.data_a ({6{1'b1}}),
				.data_b (1'b1),
```

```
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_a (1'b0),
                                .wren_b (1'b0));
        defparam
                altsyncram_component.address_aclr_a = "NONE",
                altsyncram_component.clock_enable_input_a =
"BYPASS",
                altsyncram_component.clock_enable_output_a =
"BYPASS",
                altsyncram_component.init_file = "boom.mif",
                altsyncram_component.intended_device_family =
"Cyclone V",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 1024,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a =
"UNREGISTERED",
                altsyncram_component.widthad_a = 10,
                altsyncram_component.width_a = 6,
                altsyncram_component.width_byteena_a = 1;



endmodule
```

```
//
============================================================
=
// CNX file retrieval info
//
============================================================
=
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC
"0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC
"0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING
"PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "boom.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "1024"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
```

```
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX
STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "10"
// Retrieval info: PRIVATE: WidthData NUMERIC "6"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "boom.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING
"ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "1024"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING
"UNREGISTERED"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "10"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "6"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 10 0 INPUT
NODEFVAL "address[9..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC
"clock"
// Retrieval info: USED_PORT: q 0 0 6 0 OUTPUT NODEFVAL
"q[5..0]"
// Retrieval info: CONNECT: @address_a 0 0 10 0 address 0 0
10 0
```

```
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 6 0 @q_a 0 0 6 0
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom_inst.v
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL boom_rom_bb.v
FALSE
// Retrieval info: LIB_FILE: altera_mf
```

Appendix C: Background ROM

```
// megafunction wizard:
%ROM: 1-PORT%
                                // GENERATION: STANDARD
                                // VERSION: WM1.0
                                // MODULE: altsyncram


                                //
                                ============================================================
                                // File Name: race_track_rom.v
                                // Megafunction Name(s):
                                //                    altsyncram
                                //
                                // Simulation Library Files(s):
                                //                    altera_mf
                                //
                                ============================================================
                                //
                                ************************************************************
                                // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
                                //
                                // 18.0.0 Build 614 04/24/2018 SJ Lite Edition
                                //
                                ************************************************************




                                //Copyright (C) 2018  Intel Corporation. All rights reserved.
                                //Your use of Intel Corporation's design tools, logic
                                functions
                                //and other software and tools, and its AMPP partner logic
                                //functions, and any output files from any of the foregoing
                                //(including device programming or simulation files), and any
```

```
//associated documentation or information are expressly
subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License
Agreement,
//the Intel FPGA IP License Agreement, or other applicable
license
//agreement, including, without limitation, that your use is
for
//the sole purpose of programming logic devices manufactured
by
//Intel and sold by Intel or its authorized distributors.
Please
//refer to the applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module race_track_rom (
	address,
	clock,
	q);

	input	[16:0]	address;
	input		clock;
	output	[5:0]	q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
	tri1		clock;
`ifndef ALTERA_RESERVED_QIS
```

```
// synopsys translate_on
`endif


        wire [5:0] sub_wire0;
        wire [5:0] q = sub_wire0[5:0];


        altsyncram     altsyncram_component (
                                .address_a (address),
                                .clock0 (clock),
                                .q_a (sub_wire0),
                                .aclr0 (1'b0),
                                .aclr1 (1'b0),
                                .address_b (1'b1),
                                .addressstall_a (1'b0),
                                .addressstall_b (1'b0),
                                .byteena_a (1'b1),
                                .byteena_b (1'b1),
                                .clock1 (1'b1),
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_a ({6{1'b1}}),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_a (1'b0),
```

```
                                .wren_b (1'b0));
        defparam
                altsyncram_component.address_aclr_a = "NONE",
                altsyncram_component.clock_enable_input_a =
"BYPASS",
                altsyncram_component.clock_enable_output_a =
"BYPASS",
                altsyncram_component.init_file =
"race_track.mif",
                altsyncram_component.intended_device_family =
"Cyclone V",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 76800,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a =
"UNREGISTERED",
                altsyncram_component.widthad_a = 17,
                altsyncram_component.width_a = 6,
                altsyncram_component.width_byteena_a = 1;



endmodule


//
============================================================
// CNX file retrieval info
//
============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
```

```
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING
"race_track.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "76800"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING
"0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "17"
// Retrieval info: PRIVATE: WidthData NUMERIC "6"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
```

```
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING
"../Desktop/Final 241/race_track.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING
"ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "76800"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING
"UNREGISTERED"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "17"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "6"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 17 0 INPUT NODEFVAL
"address[16..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: q 0 0 6 0 OUTPUT NODEFVAL
"q[5..0]"
// Retrieval info: CONNECT: @address_a 0 0 17 0 address 0 0
17 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 6 0 @q_a 0 0 6 0
// Retrieval info: GEN_FILE: TYPE_NORMAL race_track_rom.v
TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL race_track_rom.inc
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL race_track_rom.cmp
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL race_track_rom.bsf
FALSE
```

```
// Retrieval info: GEN_FILE: TYPE_NORMAL
race_track_rom_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL race_track_rom_bb.v
FALSE
// Retrieval info: LIB_FILE: altera_mf
```

Appendix D: Win Screen ROM

```
// megafunction wizard:
%ROM: 1-PORT%
                              // GENERATION: STANDARD
                              // VERSION: WM1.0
                              // MODULE: altsyncram


                              //
                              ============================================================
                              // File Name: win_screen_rom.v
                              // Megafunction Name(s):
                              //                    altsyncram
                              //
                              // Simulation Library Files(s):
                              //                    altera_mf
                              //
                              ============================================================
                              //
                              ************************************************************
                              // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
                              //
                              // 18.0.0 Build 614 04/24/2018 SJ Lite Edition
                              //
                              ************************************************************




                              //Copyright (C) 2018  Intel Corporation. All rights reserved.
                              //Your use of Intel Corporation's design tools, logic
                              functions
                              //and other software and tools, and its AMPP partner logic
                              //functions, and any output files from any of the foregoing
                              //(including device programming or simulation files), and any
```

```
//associated documentation or information are expressly
subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License
Agreement,
//the Intel FPGA IP License Agreement, or other applicable
license
//agreement, including, without limitation, that your use is
for
//the sole purpose of programming logic devices manufactured
by
//Intel and sold by Intel or its authorized distributors.
Please
//refer to the applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module win_screen_rom (
        address,
        clock,
        q);

        input   [16:0]  address;
        input     clock;
        output  [5:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
        tri1      clock;
`ifndef ALTERA_RESERVED_QIS
```

```
// synopsys translate_on
`endif


        wire [5:0] sub_wire0;
        wire [5:0] q = sub_wire0[5:0];


        altsyncram     altsyncram_component (
                                .address_a (address),
                                .clock0 (clock),
                                .q_a (sub_wire0),
                                .aclr0 (1'b0),
                                .aclr1 (1'b0),
                                .address_b (1'b1),
                                .addressstall_a (1'b0),
                                .addressstall_b (1'b0),
                                .byteena_a (1'b1),
                                .byteena_b (1'b1),
                                .clock1 (1'b1),
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_a ({6{1'b1}}),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_a (1'b0),
```

```
                                .wren_b (1'b0));
        defparam
                altsyncram_component.address_aclr_a = "NONE",
                altsyncram_component.clock_enable_input_a =
"BYPASS",
                altsyncram_component.clock_enable_output_a =
"BYPASS",
                altsyncram_component.init_file =
"win_screen.mif",
                altsyncram_component.intended_device_family =
"Cyclone V",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 76800,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a =
"UNREGISTERED",
                altsyncram_component.widthad_a = 17,
                altsyncram_component.width_a = 6,
                altsyncram_component.width_byteena_a = 1;



endmodule


//
============================================================
// CNX file retrieval info
//
============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
```

```
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../Desktop/NEW
PROJECT 241/win_screen.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "76800"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING
"0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "17"
// Retrieval info: PRIVATE: WidthData NUMERIC "6"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
```

```
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../Desktop/NEW
PROJECT 241/win_screen.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING
"ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "76800"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING
"UNREGISTERED"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "17"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "6"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 17 0 INPUT NODEFVAL
"address[16..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: q 0 0 6 0 OUTPUT NODEFVAL
"q[5..0]"
// Retrieval info: CONNECT: @address_a 0 0 17 0 address 0 0 17
0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 6 0 @q_a 0 0 6 0
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom.inc
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom.cmp
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom.bsf
FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom_inst.v
FALSE
```

```
// Retrieval info: GEN_FILE: TYPE_NORMAL win_screen_rom_bb.v
FALSE
// Retrieval info: LIB_FILE: altera_mf
```

Appendix E: Start Screen ROM

```
// megafunction wizard:
%ROM: 1-PORT%
                                // GENERATION: STANDARD
                                // VERSION: WM1.0
                                // MODULE: altsyncram


                                //
                                ============================================================
                                // File Name: start_screen_rom.v
                                // Megafunction Name(s):
                                //                    altsyncram
                                //
                                // Simulation Library Files(s):
                                //                    altera_mf
                                //
                                ============================================================
                                //
                                ************************************************************
                                // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
                                //
                                // 18.0.0 Build 614 04/24/2018 SJ Lite Edition
                                //
                                ************************************************************




                                //Copyright (C) 2018  Intel Corporation. All rights
                                reserved.
                                //Your use of Intel Corporation's design tools, logic
                                functions
                                //and other software and tools, and its AMPP partner logic
                                //functions, and any output files from any of the foregoing
```

```
//(including device programming or simulation files), and
any
//associated documentation or information are expressly
subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License
Agreement,
//the Intel FPGA IP License Agreement, or other applicable
license
//agreement, including, without limitation, that your use is
for
//the sole purpose of programming logic devices manufactured
by
//Intel and sold by Intel or its authorized distributors.
Please
//refer to the applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module start_screen_rom (
        address,
        clock,
        q);

        input  [16:0]  address;
        input    clock;
        output [5:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
```

```
        tri1      clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

        wire [5:0] sub_wire0;
        wire [5:0] q = sub_wire0[5:0];

        altsyncram     altsyncram_component (
                                .address_a (address),
                                .clock0 (clock),
                                .q_a (sub_wire0),
                                .aclr0 (1'b0),
                                .aclr1 (1'b0),
                                .address_b (1'b1),
                                .addressstall_a (1'b0),
                                .addressstall_b (1'b0),
                                .byteena_a (1'b1),
                                .byteena_b (1'b1),
                                .clock1 (1'b1),
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_a ({6{1'b1}}),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
```

```
                                .rden_b (1'b1),
                                .wren_a (1'b0),
                                .wren_b (1'b0));
        defparam
                altsyncram_component.address_aclr_a = "NONE",
                altsyncram_component.clock_enable_input_a =
"BYPASS",
                altsyncram_component.clock_enable_output_a =
"BYPASS",
                altsyncram_component.init_file =
"start_screen.mif",
                altsyncram_component.intended_device_family =
"Cyclone V",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 76800,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a =
"UNREGISTERED",
                altsyncram_component.widthad_a = 17,
                altsyncram_component.width_a = 6,
                altsyncram_component.width_byteena_a = 1;



endmodule


//
============================================================
// CNX file retrieval info
//
============================================================
```

```
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC
"0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING
"../Desktop/start_screen.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "76800"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING
"0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "17"
// Retrieval info: PRIVATE: WidthData NUMERIC "6"
// Retrieval info: PRIVATE: rden NUMERIC "0"
```

```
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING
"BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING
"../Desktop/start_screen.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING
"Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING
"ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "76800"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING
"UNREGISTERED"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "17"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "6"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 17 0 INPUT
NODEFVAL "address[16..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC
"clock"
// Retrieval info: USED_PORT: q 0 0 6 0 OUTPUT NODEFVAL
"q[5..0]"
// Retrieval info: CONNECT: @address_a 0 0 17 0 address 0 0
17 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 6 0 @q_a 0 0 6 0
// Retrieval info: GEN_FILE: TYPE_NORMAL start_screen_rom.v
TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL
start_screen_rom.inc FALSE
```

```
// Retrieval info: GEN_FILE: TYPE_NORMAL
start_screen_rom.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL
start_screen_rom.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL
start_screen_rom_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL
start_screen_rom_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```

Appendix F: Delay Counter

```
module
DelayCoun
ter(

        //Calls DelayCounter and feeds in a large count down number, clock,
simreset-----------------, and OneFrameCounter--------------------
        input Clock, simReset,
        input[19:0] countDownNum,
        output reg[19:0] RDOut);

        //with every iteration of posedge clock
        always @ (posedge Clock)
                begin

                        //if simReset is true, then set RDOut to 0
                        if (simReset)
                                RDOut <= 20'd0;

                        //if RDOut is 0, set RDOut to the countdown number
                        else if (RDOut == 20'd0)
                                RDOut <= countDownNum;
                        //if simreset is false, and RDOut is not 0, then
decrease RDOut by 1
                        else
                                RDOut <= RDOut - 20'd1;
                end

endmodule
```

Appendix G: Seconds Counter

```
module
SecondsCounter(
                                input Clock, simReset,
                                input[27:0] countDownNum,
                                output reg[27:0] RDOut);

                                always @ (posedge Clock)
                                        begin
                                                if (simReset)
                                                        RDOut <= 28'd0;
                                                else if (RDOut == 28'd0)
                                                        RDOut <= countDownNum;
                                                else
                                                        RDOut <= RDOut - 28'd1;
                                        end

                        endmodule
```

Appendix H: Datapath

```
module
datapath
(

        //initialize inputs for clock, reset and moving in 3 directions
        input Clock, Resetn, moveForward, moveRight, moveLeft,

        //initialize inputs for setting reset signals, starting the race,
drawing the background, car and over the car
        input setResetSignals, startRace, drawBG, drawCar, drawErase,

        //initialize inputs for moving, drawing explosion and the start/win
screen
        input move, drawBoom, drawStartScreen, drawWinScreen,

        //initialize output for checking if the background, car, drawover car
have been drawn and if the race is finished
        output reg DoneDrawBG, DoneDrawCar, DoneDrawErase, FinishedRace,

        //initalize output that checks for collision, whether the explosion,
start screen and win screen has been draw
        output reg Collision, DoneDrawBoom, DoneDrawStartScreen,
DoneDrawWinScreen,

        //initalize output for the colour, x position and y position to be
displayed
        output reg[5:0] colourDisplay,
        output reg[7:0] yDisplay,
        output reg[8:0] xDisplay);

        //---------------------------------RAM, Registers, and
Wires---------------------------------
```

```
//sets local paramaters
localparam orientRight          = 0,
                        orientUpRightRight   = 1,
                    orientUpRight      = 2,
                        orientUpUpRight       = 3,
                    orientUp           = 4,
                        orientUpUpLeft               = 5,
                        orientUpLeft          = 6,
                        orientUpLeftLeft      = 7,
                    orientLeft         = 8,
                        orientDownLeftLeft   = 9,
                    orientDownLeft     = 10,
                        orientDownDownLeft   = 11,
                    orientDown         = 12,
                        orientDownDownRight  = 13,
                    orientDownRight    = 14,
                        orientDownRightRight = 15;


//addresses for ROM files
reg[13:0] carAddress = 14'd0;
reg[16:0] backgroundAddress = 17'd0;
reg[16:0] startScreenAddress = 17'd0;
reg[16:0] winScreenAddress = 17'd0;
reg[9:0] boomAddress = 10'd0;


//bits used for the ROM files that are used in displaying pixels
wire[5:0] carColourToDisplay;
```

```
wire[5:0] backgroundColourToDisplay;
wire[5:0] boomColourToDisplay;
wire[5:0] winScreenColourToDisplay;
wire[5:0] startScreenColourToDisplay;



//only being used to read the rom files, not write
car_rom carRom(
        .address(carAddress),
        .clock(Clock),
        .q(carColourToDisplay));

race_track_rom track(
        .address(backgroundAddress),
        .clock(Clock),
        .q(backgroundColourToDisplay));

boom_rom boom(
        .address(boomAddress),
        .clock(Clock),
        .q(boomColourToDisplay));


win_screen_rom win(
        .address(winScreenAddress),
        .clock(Clock),
        .q(winScreenColourToDisplay));


start_screen_rom start(
        .address(startScreenAddress),
```

```
                .clock(Clock),
                .q(startScreenColourToDisplay));


        //creates variables used for checking if the car past the finished line
        //creates storage variables for the x and y position as well as counters
for the positions
        //creates storage variable for the orientation and how many pixels are
to be moved
        reg pastStartLine = 1'b0;
        reg[7:0] yCount = 8'd0;
        reg[7:0] currentYPosition = 8'd0;
        reg[8:0] xCount = 9'd0;
        reg[8:0] currentXPosition = 9'd0;
        reg[3:0] currentOrientation = orientRight;
        localparam pixelsToMove = 2;

        //---------------------------------Operations Based on
Input---------------------------------


        //occurs at every iteration of the posedge clock
        always@(posedge Clock) begin

                //-----------------------------------Resetting
Signals-----------------------------------


                //if resetting signals
                if(setResetSignals) begin

                        //sets all the values for the storage variables to 0
```

```
                backgroundAddress <= 17'd0;
                carAddress <= 14'd0;
                boomAddress <= 10'd0;
                startScreenAddress <= 17'd0;
                winScreenAddress <= 17'd0;
                currentXPosition <= 9'd0;
                currentYPosition <= 8'd0;
                xCount <= 9'd0;
                yCount <= 8'd0;
                DoneDrawBG <= 1'b0;
                DoneDrawCar <= 1'b0;
                DoneDrawBoom <= 1'b0;
                DoneDrawErase <= 1'b0;
                Collision <= 1'b0;
                currentOrientation <= orientRight;
                FinishedRace <= 1'b1;
                pastStartLine <= 1'b0;
                DoneDrawStartScreen <=1'b0;
                DoneDrawWinScreen <= 1'b0;
            end

            //if the game has started, then set FinishedRace to 0 since the
game is not finished
            if(startRace) FinishedRace <= 1'b0;


            //-----------------------------------Drawing Start
Screen----------------------------------

            //while the start screen is being drawn
            if(drawStartScreen && !DoneDrawStartScreen)
```

```
		begin

			//set the colour to be displayed as the one read from the rom file of the start screen
			//while moving through the image
			colourDisplay <= startScreenColourToDisplay;
			xDisplay <= currentXPosition + xCount;
			yDisplay <= currentYPosition + yCount;

			//if the rom file reaches the end, then reset the addresses and set DoneDrawStartScreen to true
			if(xCount == 9'd319 && yCount == 8'd239)
			begin
				xCount <= 9'd0;
				yCount <= 8'd0;

				currentXPosition <= 9'd0;
				currentYPosition <= 8'd0;
				startScreenAddress <= 17'd0;
				DoneDrawStartScreen <= 1'b1;
			end

			//if the rom file reaches the end of the row, then start at the beginning of the row underneath
			else if(xCount == 9'd319)
			begin
				xCount <= 9'd0;
				yCount <= yCount + 8'd1;
				startScreenAddress <= startScreenAddress + 17'd1;
				DoneDrawStartScreen <= 1'b0;
			end
```

```
                        //moves through the row of the rom
                        else begin
                                xCount <= xCount + 9'd1;
                                startScreenAddress <= startScreenAddress + 17'd1;
                                DoneDrawStartScreen <= 1'b0;
                        end
                end



                //-----------------------------------Drawing Win
Screen-----------------------------------

                //while drawing the win screen
                if(drawWinScreen && !DoneDrawWinScreen)
                begin

                        //set the colour to be displayed as the one read from the
rom file of the start screen
                        //while moving through the image
                        colourDisplay <= winScreenColourToDisplay;
                        xDisplay <= currentXPosition + xCount;
                        yDisplay <= currentYPosition + yCount;

                        //if the rom file reaches the end, then reset the
addresses and set DoneDrawWinScreen to true
                        if(xCount == 9'd319 && yCount == 8'd239)
                        begin
                                xCount <= 9'd0;
                                yCount <= 8'd0;
```

```
                        currentXPosition <= 9'd0;
                        currentYPosition <= 8'd0;
                        winScreenAddress <= 17'd0;
                        DoneDrawWinScreen <= 1'b1;
                    end

                    //if the rom file reaches the end of the row, then start
at the beginning of the row underneath
                    else if(xCount == 9'd319)
                    begin
                        xCount <= 9'd0;
                        yCount <= yCount + 8'd1;
                        winScreenAddress <= winScreenAddress + 17'd1;
                        DoneDrawWinScreen <= 1'b0;
                    end

                    //moves through the row of the rom
                    else begin
                        xCount <= xCount + 9'd1;
                        winScreenAddress <= winScreenAddress + 17'd1;
                        DoneDrawWinScreen <= 1'b0;
                    end
                end


                //-----------------------------------Drawing
Background-----------------------------------

                //while drawing the background
                if(drawBG && !DoneDrawBG) begin
```

```
                    //set the colour to be displayed as the one read from the
rom file of the start screen
                    //while moving through the image
                    colourDisplay <= backgroundColourToDisplay;
                    xDisplay <= currentXPosition + xCount;
                    yDisplay <= currentYPosition + yCount;


                    //if the rom file reaches the end, then reset the
addresses and set DoneDrawBG to true
                    //after drawing the background, sets the x and y position
to that of the car's position
                    if(xCount == 9'd319 && yCount == 8'd239)
                    begin
                            xCount <= 9'd0;
                            yCount <= 8'd0;

                            currentXPosition <= 9'd94;
                            currentYPosition <= 8'd193;
                            backgroundAddress <= (320 * 193) + 94;

                            DoneDrawBG <= 1'b1;
                    end

                    //if the rom file reaches the end of the row, then start
at the beginning of the row underneath
                    else if(xCount == 9'd319)
                    begin
                            xCount <= 9'd0;
                            yCount <= yCount + 8'd1;
                            backgroundAddress <= backgroundAddress + 17'd1;
                            DoneDrawBG <= 1'b0;
```

```
                        end

                        //moves through the row of the rom
                        else begin
                                xCount <= xCount + 9'd1;
                                backgroundAddress <= backgroundAddress + 17'd1;
                                DoneDrawBG <= 1'b0;
                        end
                end

                //--------------------------------------Drawing
Car--------------------------------------


                //while drawing car
                else if(drawCar && !DoneDrawCar) begin


                        //to erase the background in the rom file, detect given
colour and plot background at those pixels
                        if(carColourToDisplay == 6'b100010) begin
                                colourDisplay <= backgroundColourToDisplay;
                                xDisplay <= currentXPosition + xCount;
                                yDisplay <= currentYPosition + yCount;
                        end

                        //draws the car
                        else begin
                                colourDisplay <= carColourToDisplay; // Colour of
sprite at current position
                                xDisplay <= currentXPosition + xCount;
                                yDisplay <= currentYPosition + yCount;
```

```
                        end

                        //after drawing the car, reset counter and restore
address to the top left corner of the box
                        if(xCount == 9'd31 && yCount == 8'd31) begin
                                xCount <= 9'd0;
                                yCount <= 8'd0;
                                backgroundAddress <= backgroundAddress + (-(320 *
31) - 31);
                                DoneDrawCar <= 1'b1;
                        end

                        //if the rom file reaches the end of the row, then start
at the beginning of the row underneath
                        else if(xCount == 9'd31) begin
                                xCount <= 9'd0;
                                yCount <= yCount + 8'd1;
                                carAddress <= carAddress + 14'd1;
                                backgroundAddress <= backgroundAddress + (320 -
31);
                                DoneDrawCar <= 1'b0;
                        end

                        //moves through the row of the rom
                        else begin
                                xCount <= xCount + 9'd1;
                                carAddress <= carAddress + 14'd1;
                                backgroundAddress <= backgroundAddress + 17'd1;
                                DoneDrawCar <= 1'b0;
                        end

                        //checking if the car address is not purple at a given
location
```

```
                        //and background is green

                        if(backgroundColourToDisplay == 6'b001001 &&
carColourToDisplay !=  6'b100010) begin
                                DoneDrawCar <= 1'b1;
                                Collision <= 1'b1;
                                backgroundAddress <= backgroundAddress + (-(320 *
yCount) - xCount);
                                xCount <= 9'd0;
                                yCount <= 8'd0;


                        end
                end

                //---------------------------------------Drawing
Explosion---------------------------------------

                //while drawing the explosion
                if(drawBoom && !DoneDrawBoom) begin


                        //outputs the explosion display
                        colourDisplay <= boomColourToDisplay;
                        xDisplay <= currentXPosition + xCount;
                        yDisplay <= currentYPosition + yCount;


                        //resets signals when explosion has been drawn
                        if(xCount == 9'd31 && yCount == 8'd31) begin
                                xCount <= 9'd0;
                                yCount <= 8'd0;
```

```
                        backgroundAddress <= backgroundAddress + (-(320 *
31) - 31);
                        DoneDrawBoom <= 1'b1;
                        FinishedRace <= 1'b1;
                    end

                    //if the rom file reaches the end of the row, then start
at the beginning of the row underneath
                    //and goes to the next row of the background as well
keeping the dimensions of the rom in mind (-31)
                    else if(xCount == 9'd31) begin
                        xCount <= 9'd0;
                        yCount <= yCount + 8'd1;
                        boomAddress <= boomAddress + 10'd1;
                        backgroundAddress <= backgroundAddress + (320 -
31);
                        DoneDrawBoom <= 1'b0;
                    end

                    //moves through the row of the rom
                    else begin
                        xCount <= xCount + 9'd1;
                        boomAddress <= boomAddress + 10'd1;
                        backgroundAddress <= backgroundAddress + 17'd1;
                        DoneDrawBoom <= 1'b0;
                    end
                end

                //while drawing over the car
                if(drawErase && !DoneDrawErase) begin

                    //erase the car and replace with background display
```

```
                        colourDisplay <= backgroundColourToDisplay;
                        xDisplay <= currentXPosition + xCount;
                        yDisplay <= currentYPosition + yCount;

                        //once the car has been erased, reset count and restore
address to top left corner of car box
                        if(xCount == 9'd31 && yCount == 8'd31) begin
                                xCount <= 9'd0;
                                yCount <= 8'd0;
                                backgroundAddress <= backgroundAddress + (-(320 *
31) - 31);
                                DoneDrawErase <= 1'b1;
                        end

                        //if the rom file reaches the end of the row, then start
at the beginning of the row underneath
                        //and goes to the next row of the background as well
keeping the dimensions of the rom in mind (-31)
                        else if(xCount == 9'd31) begin
                                xCount <= 9'd0;
                                yCount <= yCount + 8'd1;
                                backgroundAddress <= backgroundAddress + (320 -
31);
                                DoneDrawErase <= 1'b0;
                        end

                        //moves through the row of the rom
                        else begin
                                xCount <= xCount + 9'd1;
                                backgroundAddress <= backgroundAddress + 17'd1;
                                DoneDrawErase <= 1'b0;
                        end
                end
```

```
                //--------------------------------------Moving
Car--------------------------------------

                // If move is true, then enter this if statement
                if(move) begin

                        // Signals from drawing are reset to prepare for the next
draw
                        DoneDrawBG <= 1'b0;
                        DoneDrawCar <= 1'b0;
                        DoneDrawErase <= 1'b0;
                        DoneDrawWinScreen <= 1'b0;
                        DoneDrawStartScreen <= 1'b0;


                        // enter this case regardless of what orientation the car
is in
                        case(currentOrientation)
                                // if the car is pointing to the right
                                orientRight: begin
                                        // if the user wants to move forward
                                        if(moveForward) begin
                                                carAddress <= orientRight * 1024;
                                                // orientation doesn't change if the
user moves forward -- it is still right
                                                currentOrientation <= orientRight;
                                                backgroundAddress <=
backgroundAddress + pixelsToMove;
                                                // Car moves right by 2 pixels, only
x position changes
                                                currentXPosition <= currentXPosition
+ pixelsToMove;
                                        end
```

```
                                        // if user presses right
                                        else if(moveRight) begin
                                                // Moves to a different sprite
                                                carAddress <= orientDownRightRight *
1024;
                                                // orientation changed to -45
degrees
                                                currentOrientation <=
orientDownRightRight;
                                                // Car stays at the same location,
no need to change x and y position
                                        end
                                        else if (moveLeft) begin
                                                carAddress <= orientUpRightRight *
1024; // Car stays at the same location
                                                // orientation changes to 45 degrees
                                                currentOrientation <=
orientUpRightRight;
                                        end
                                end

                                // orientation when car is at 22.5 degrees
                                orientUpRightRight: begin
                                        // if the user presses the forward key
                                        if(moveForward) begin
                                                carAddress <= orientUpRightRight *
1024;
                                                // orienation stays the same
                                                currentOrientation <=
orientUpRightRight;
                                                backgroundAddress <=
backgroundAddress + (-(320 * (pixelsToMove - 1)) + pixelsToMove);
                                                currentXPosition <= currentXPosition
+ pixelsToMove;
                                                currentYPosition <= currentYPosition
- (pixelsToMove - 1);
```

```
                                end
                                // if the user presses the right key
                                else if(moveRight) begin
                                        carAddress <= orientRight * 1024;
                                        // orientation reverts back to 0
degrees
                                        currentOrientation <= orientRight;
                                end
                                // if user presses the left key
                                else if (moveLeft) begin
                                        carAddress <= orientUpRight * 1024;
                                        // orientation is 45 degrees
                                        currentOrientation <= orientUpRight;
                                end
                        end

                        // orientation when car is 45 degrees
                        orientUpRight: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientUpRight * 1024;
                                        // orientation is 45 degrees
                                        currentOrientation <= orientUpRight;
                                        backgroundAddress <=
backgroundAddress + (-(320 * pixelsToMove) + pixelsToMove);
                                        // x and y both change at constant
rate
                                        currentXPosition <= currentXPosition
+ pixelsToMove;
                                        currentYPosition <= currentYPosition
- pixelsToMove;
                                end
                                // if user presses right
```

```
                                else if(moveRight) begin
                                        carAddress <= orientUpRightRight *
1024;
                                        // orientation is 22.5 degrees
                                        currentOrientation <=
orientUpRightRight;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientUpUpRight *
1024;
                                        // orientation is 67.5 degrees
                                        currentOrientation <=
orientUpUpRight;
                                end
                        end
                        // orientation of car is 67.5 degrees
                        orientUpUpRight: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientUpUpRight *
1024;
                                        // orientation remains the same
                                        currentOrientation <=
orientUpUpRight;
                                        backgroundAddress <=
backgroundAddress + (-(320 * pixelsToMove) + (pixelsToMove - 1));
                                        // x position changes slightly
                                        currentXPosition <= currentXPosition
+ (pixelsToMove - 1);
                                        // car goes up, so pixels get
subtracted from the y position
                                        currentYPosition <= currentYPosition
- pixelsToMove;
                                end
```

```
                else if(moveRight) begin
                        carAddress <= orientUpRight * 1024;
                        currentOrientation <= orientUpRight;
                end
                else if(moveLeft) begin
                        carAddress <= orientUp * 1024;
                        currentOrientation <= orientUp;
                end
        end

        // orientation when car is 90 degrees
        orientUp: begin
                // if user presses forward
                if(moveForward) begin
                        carAddress <= orientUp * 1024;
                        // orientation stays the same
                        currentOrientation <= orientUp;
                        backgroundAddress <=
backgroundAddress - (320 * pixelsToMove);
                        // car goes up, so pixels get
subtracted from the y position
                        currentYPosition <= currentYPosition
- pixelsToMove;
                end
                // if user presses right
                else if(moveRight) begin
                        carAddress <= orientUpUpRight *
1024;
                        // orientation is 67.5 degrees
                        currentOrientation <=
orientUpUpRight;
                end
                // if user presses left
```

```
                                        else if(moveLeft) begin
                                                carAddress <= orientUpUpLeft * 1024;
                                                // orientation is 112.5 degrees
                                                currentOrientation <=
orientUpUpLeft;
                                        end
                                end
                                // orientation is 112.5 degrees
                                orientUpUpLeft: begin
                                        // if user presses forward
                                        if(moveForward) begin
                                                carAddress <= orientUpUpLeft * 1024;
                                                // orientation remains the same
                                                currentOrientation <=
orientUpUpLeft;
                                                backgroundAddress <=
backgroundAddress + (-(320 * pixelsToMove) - (pixelsToMove - 1));
                                                // x position changes slightly
                                                currentXPosition <= currentXPosition
- (pixelsToMove - 1);
                                                // car goes up, so pixels get
subtracted from the y position
                                                currentYPosition <= currentYPosition
- pixelsToMove;
                                        end
                                        // if user presses right
                                        else if(moveRight) begin
                                                carAddress <= orientUp * 1024;
                                                // orientation is 90 degrees
                                                currentOrientation <= orientUp;
                                        end
                                        // if user presses left
                                        else if(moveLeft) begin
```

```
                                        carAddress <= orientUpLeft * 1024;
                                        // orientation is 135 degrees
                                        currentOrientation <= orientUpLeft;
                                end
                        end
                        // orientation is 135 degrees
                        orientUpLeft: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientUpLeft * 1024;
                                        // orientation stays the same
                                        currentOrientation <= orientUpLeft;
                                        backgroundAddress <=
backgroundAddress + (-(320 * pixelsToMove) - pixelsToMove);
                                        // x and y positions both decrease
at same rate
                                        currentXPosition <= currentXPosition
- pixelsToMove;
                                        currentYPosition <= currentYPosition
- pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientUpUpLeft * 1024;
                                        // orientation is 112.5 degrees
                                        currentOrientation <=
orientUpUpLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientUpLeftLeft *
1024;
                                        // orientation is 157.5 degrees
```

```
                                        currentOrientation <=
orientUpLeftLeft;
                                end
                        end
                        // orientation is 157.5 degrees
                        orientUpLeftLeft: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientUpLeftLeft *
1024;
                                        // orientation is the same
                                        currentOrientation <=
orientUpLeftLeft;
                                        backgroundAddress <=
backgroundAddress + (-(320 * (pixelsToMove - 1)) - pixelsToMove);
                                        // y changes slightly, x moves at
the regular rate
                                        currentXPosition <= currentXPosition
- pixelsToMove;
                                        currentYPosition <= currentYPosition
- (pixelsToMove - 1);
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientUpLeft * 1024;
                                        // oritentation is 135 degrees
                                        currentOrientation <= orientUpLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientLeft * 1024;
                                        // orientation is 180 degrees
                                        currentOrientation <= orientLeft;
                                end
```

```
                        end
                        // orientation is 180 degrees
                        orientLeft: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientLeft * 1024;
                                        currentOrientation <= orientLeft;
                                        backgroundAddress <=
backgroundAddress - pixelsToMove;
                                        // only x changes at constant rate
                                        currentXPosition <= currentXPosition
- pixelsToMove;
                                end
                                // if user pressses  right
                                else if(moveRight) begin
                                        carAddress <= orientUpLeftLeft *
1024;
                                        // orientation is 157.5 degrees
                                        currentOrientation <=
orientUpLeftLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDownLeftLeft *
1024;
                                        // orientation is 202.5 degrees
                                        currentOrientation <=
orientDownLeftLeft;
                                end
                        end
                        // orientation is 202.5 degrees
                        orientDownLeftLeft: begin
                                // if user presses forward
```

```
                                        if(moveForward) begin
                                                carAddress <= orientDownLeftLeft *
1024;
                                                // orientation remains the same
                                                currentOrientation <=
orientDownLeftLeft;
                                                backgroundAddress <=
backgroundAddress + (320 * (pixelsToMove - 1) - pixelsToMove);
                                                // x changes constantly, y changes
slightly
                                                currentXPosition <= currentXPosition
- pixelsToMove;
                                                currentYPosition <= currentYPosition
+ (pixelsToMove - 1);
                                        end
                                        // if user presses right
                                        else if(moveRight) begin
                                                carAddress <= orientLeft * 1024;
                                                // orientation is 180 degrees
                                                currentOrientation <= orientLeft;
                                        end
                                        // if user presses left
                                        else if(moveLeft) begin
                                                carAddress <= orientDownLeft * 1024;
                                                // orientation is 247.5 degrees
                                                currentOrientation <=
orientDownLeft;
                                        end
                                end
                                // orientation is 247.5 degrees
                                orientDownLeft: begin
                                        // if user presses forward
                                        if(moveForward) begin
                                                carAddress <= orientDownLeft * 1024;
```

```
                                        // orientation remains the same
                                        currentOrientation <=
orientDownLeft;
                                        backgroundAddress <=
backgroundAddress + ((320 * pixelsToMove) - pixelsToMove);
                                        // y and x both change constantly,
but y increases since the car goes down
                                        currentXPosition <= currentXPosition
- pixelsToMove;
                                        currentYPosition <= currentYPosition
+ pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientDownLeftLeft *
1024;
                                        // orientation is 202.5 degrees
                                        currentOrientation <=
orientDownLeftLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDownDownLeft *
1024;
                                        // orientation is 247.5 degrees
                                        currentOrientation <=
orientDownDownLeft;
                                end
                        end
                        // orientation is 247.5 degrees
                        orientDownDownLeft: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientDownDownLeft *
1024;
```

```
                                        // orientation is 247.5 degrees
                                        currentOrientation <=
orientDownDownLeft;
                                        backgroundAddress <=
backgroundAddress + (320 * pixelsToMove - (pixelsToMove - 1));
                                        // y changes constantly but x moves
at a slower rate
                                        currentXPosition <= currentXPosition
- (pixelsToMove - 1);
                                        currentYPosition <= currentYPosition
+ pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientDownLeft * 1024;
                                        // orientation is 225 degrees
                                        currentOrientation <=
orientDownLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDown * 1024;
                                        // orientation is 270 degrees
                                        currentOrientation <= orientDown;
                                end
                        end
                        // orientation is 270 degrees
                        orientDown: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientDown * 1024;
                                        // orientation stays the same
                                        currentOrientation <= orientDown;
```

```
                                        backgroundAddress <=
backgroundAddress + (320 * pixelsToMove);
                                        // only y changes at a constant rate
                                        currentYPosition <= currentYPosition
+ pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientDownDownLeft *
1024;
                                        // orientation is 247.5 degrees
                                        currentOrientation <=
orientDownDownLeft;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDownDownRight *
1024;
                                        // orientation is 292.5 degrees
                                        currentOrientation <=
orientDownDownRight;
                                end
                        end
                        // orientation is 292.5 degrees
                        orientDownDownRight: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientDownDownRight *
1024;
                                        // orientation stays the same
                                        currentOrientation <=
orientDownDownRight;
                                        backgroundAddress <=
backgroundAddress + (320 * pixelsToMove + (pixelsToMove - 1));
```

```
                                        // x changes at a slower rate but y
changes at the constant rate
                                        currentXPosition <= currentXPosition
+ (pixelsToMove - 1);
                                        currentYPosition <= currentYPosition
+ pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientDown * 1024;
                                        // orientation is 270 degrees
                                        currentOrientation <= orientDown;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDownRight *
1024;
                                        // orientation is 315 degrees
                                        currentOrientation <=
orientDownRight;
                                end
                        end
                        // orientation is 315 degrees
                        orientDownRight: begin
                                // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientDownRight *
1024;
                                        // orientation stays the same
                                        currentOrientation <=
orientDownRight;
                                        backgroundAddress <=
backgroundAddress + ((320 * pixelsToMove) + pixelsToMove);
                                        // x and y move at constant rate
```

```
                                        currentXPosition <= currentXPosition
+ pixelsToMove;
                                        currentYPosition <= currentYPosition
+ pixelsToMove;
                                end
                                // if user presses right
                                else if(moveRight) begin
                                        carAddress <= orientDownDownRight *
1024;
                                        // orientation is 292.5 degrees
                                        currentOrientation <=
orientDownDownRight;
                                end
                                // if user presses left
                                else if(moveLeft) begin
                                        carAddress <= orientDownRightRight *
1024;
                                        // orientation is 337.5 degrees
                                        currentOrientation <=
orientDownRightRight;
                                end
                        end
                        // orientation is 337.5 degrees
                        orientDownRightRight: begin
                        // if user presses forward
                                if(moveForward) begin
                                        carAddress <= orientDownRightRight *
1024;
                                        // orientation stays the same
                                        currentOrientation <=
orientDownRightRight;
                                        backgroundAddress <=
backgroundAddress + (320 * (pixelsToMove - 1) + pixelsToMove);
                                        // x changes at a constant rate, but
y moves at a slower rate
```

```
                                        currentXPosition <= currentXPosition
+ pixelsToMove;
                                        currentYPosition <= currentYPosition
+ (pixelsToMove - 1);
                                    end
                                    // if user presses right
                                    else if(moveRight) begin
                                        carAddress <= orientDownRight *
1024;
                                        // orientation is 315 degrees
                                        currentOrientation <=
orientDownRight;
                                    end
                                    // if user presses left
                                    else if(moveLeft) begin
                                        carAddress <= orientRight * 1024;
                                        // orientation is 0 degrees
                                        currentOrientation <= orientRight;
                                    end
                                end

                        endcase
                end

                // the finish line conditions, the y goes from 183 to 206
                if(currentXPosition == 9'd150 && currentYPosition > 8'd100 &&
                   currentYPosition < 8'd240)
                        pastStartLine <= 1'b1;

                //DRAWING WIN SCREEN
                if(!FinishedRace && pastStartLine && (currentXPosition == 9'd124
|| currentXPosition == 9'd125) &&
```

```
                    currentYPosition > 8'd187 && currentYPosition < 8'd199)
begin// numbers for y may change after testing on board
                                FinishedRace <= 1'b1;
                                currentXPosition <= 9'd0;
                                currentYPosition <= 8'd0;
                                xCount <= 9'd0;
                                yCount <= 8'd0;
                end
        end


endmodule
```

Appendix I: Controlpath

```
module
control
(

        //takes in input for clock, reset, EnableOneFrame, starting the game,
and directions of motion
        input Clock, Resetn, EnableOneFrame, start, forward, right, left,


        //takes in input for conditions of whether certain situations have been
drawn and completed
        input DoneDrawBG, DoneDrawCar, DoneDrawErase, DoneDrawBoom,
DoneDrawStartScreen, DoneDrawWinScreen, FinishedRace, Collision,


        //takes in storage variables for starting the race track display, and
drawing all features
        output reg setResetSignals, startRace, drawBG, drawCar, drawErase,
drawBoom, drawStartScreen, drawWinScreen, move, plot);


        //takes in storage values for current state and next state
        reg[3:0] current_state, next_state;



        //sets local parameters for the various states of the game
        localparam  DRAW_START_SCREEN = 0,
                                    START_RACE              = 1,
                                    SET_RESET_SIGNALS = 2,
                                    DRAW_BACKGROUND    = 3,
                                    DRAW_CAR           = 4,
                                    WAIT_FOR_MOVE      = 5,
                                    DRAW_OVER_CAR      = 6,
                                    MOVE_FORWARD       = 7,
```

```
                                    MOVE_LEFT_RIGHT    = 8,
                                    WAIT_LEFT_RIGHT    = 9,
                                    DRAW_EXPLOSION     = 10,
                                    DRAW_WIN_SCREEN    = 11;

    // Next state logic aka our state table
    always@(*)

         //begins the state table
    begin: state_table
                case (current_state)

                        //When drawing the start screen
                        DRAW_START_SCREEN:
                        begin
                                //if the start screen is done being drawn
                                if(DoneDrawStartScreen)
                                begin
                                        //if the game has finished and is not being
played, draw start screen next
                                        if(!start && FinishedRace) next_state =
DRAW_START_SCREEN;

                                        //if the game is to be played, start the
ract next
                                        else if(start) next_state = START_RACE;

                                        //otherwise draw start screen next
                                        else next_state = DRAW_START_SCREEN;
                                end
```

```
                        //if start screen is not done being drawn, draw
start screen next
                        else next_state = DRAW_START_SCREEN;
                end

                //When the race is ready to start, draw background next
                START_RACE: next_state = DRAW_BACKGROUND;

                //when signals have been reset, draw the start screen for
a game reset
                SET_RESET_SIGNALS: next_state = DRAW_START_SCREEN;

                //draw the background if the background has not been drawn
                //otherwise draw the car if the background has been drawn
                DRAW_BACKGROUND: next_state = DoneDrawBG ? DRAW_CAR :
DRAW_BACKGROUND;

                //when car is being drawn
                DRAW_CAR: begin

                        //if car is done being drawn
                        if(DoneDrawCar)
                        begin
                                //if the game has not started, reset
signals next
                                if(!start) next_state = SET_RESET_SIGNALS;
                                //if the race is finished, draw the win
screen next
                                else if(FinishedRace) next_state =
DRAW_WIN_SCREEN;
                                //if collision has occured, draw explosion
next
                                else if(Collision) next_state =
DRAW_EXPLOSION;
```

```
                                        //if user moves forward and EnableOneFrame
is true, draw over car next
                                        else if(forward == 1'b1 && EnableOneFrame)
next_state = DRAW_OVER_CAR;
                                        //if user turns left or right, then wait
for the left/right turn  next
                                        else if(left == 1'b1 || right == 1'b1)
next_state = WAIT_LEFT_RIGHT;
                                        //otherwise wait for move next
                                        else next_state = WAIT_FOR_MOVE;
                                end
                                //if the car is not drawn, draw the car
                                else next_state = DRAW_CAR;
                        end


                        //when waiting for a move
                        WAIT_FOR_MOVE: begin
                                //if user moves forward and EnableOneFrame is
true, draw over car next
                                if(forward == 1'b1 && EnableOneFrame) next_state =
DRAW_OVER_CAR;
                                //if user turns left or right, then wait for the
left/right turn  next
                                else if(left == 1'b1 || right == 1'b1) next_state
= DRAW_OVER_CAR;
                                //otherwise wait for a move
                                else next_state = WAIT_FOR_MOVE;
                        end


                        //when drawing over the car
                        DRAW_OVER_CAR: begin


                                //if the car has been drawn over
                                if(DoneDrawErase) begin
```

```
                                    //if moving foward, move forward next
                                    if(forward == 1'b1) next_state =
MOVE_FORWARD;
                                    //if moving left or right, move left/right
next
                                    else if(left == 1'b1 || right == 1'b1)
next_state = MOVE_LEFT_RIGHT;
                                    //otherwise draw car next
                                    else next_state = DRAW_CAR;
                            end
                            //if the car not been drawn over, draw the car
over
                            else next_state = DRAW_OVER_CAR;
                    end

                    //when moving forward, draw the car next
                    MOVE_FORWARD: next_state = DRAW_CAR;

                    //when movign left or right, draw the car next
                    MOVE_LEFT_RIGHT: next_state = DRAW_CAR;

                    //when waiting for left/right, wait for left/right again
if car turned left or right
                    //if car didnt turn left or right, then wait for move next
                    WAIT_LEFT_RIGHT: next_state = (left == 1'b1 || right ==
1'b1) ? WAIT_LEFT_RIGHT : WAIT_FOR_MOVE;

                    //when drawing the explosion
                    DRAW_EXPLOSION: begin

                            //if explosion is done being drawn
                            if(DoneDrawBoom) begin
```

```
                                        //if the game is in motion, draw explosion
next
                                        if(start) next_state = DRAW_EXPLOSION;

                                        //otherwise reset signals next since game
is not in motion
                                        else next_state = SET_RESET_SIGNALS;
                                end
                                //if explosion is not done being drawn, draw
explosion
                                else next_state = DRAW_EXPLOSION;
                        end

                        //when drawing the win screen
                        DRAW_WIN_SCREEN: begin

                                //if win screen is done being drawn
                                if(DoneDrawWinScreen) begin

                                        //and the game has started already, draw
win screen
                                        if(start) next_state = DRAW_WIN_SCREEN;

                                        //otherwise reset signals
                                        else next_state = SET_RESET_SIGNALS;
                                end
                                //if win screen is not done being drawn, draw win
screen
                                else next_state = DRAW_WIN_SCREEN;
                        end

                        //default case is to reset signals
```

```
                default: next_state = SET_RESET_SIGNALS;
            endcase
end // state_table


// Output logic aka all of our datapath control signals
always @(*)
begin: enable_signals

            //sets values for all the cases
            setResetSignals = 1'b0;
            startRace = 1'b0;
            drawBG = 1'b0;
            drawCar = 1'b0;
            drawErase = 1'b0;
            drawBoom = 1'b0;
            drawStartScreen = 1'b0;
            drawWinScreen = 1'b0;
            move = 1'b0;
            plot = 1'b0;


    case (current_state)

                //when drawing the start screen
                DRAW_START_SCREEN: begin
                        //drawing start screen and plotting are true
                        drawStartScreen = 1'b1;
                        plot = 1'b1;
                end
                //when reseting signals, setResetSignals are ture
```

```
                SET_RESET_SIGNALS: setResetSignals = 1'b1;

                //when drawing background, drawbackground and plot are
true
                DRAW_BACKGROUND: begin
                        drawBG = 1'b1;
                        plot = 1'b1;
                end

                //when starting race, startrace is true
                START_RACE: startRace = 1'b1;

                //when drawing car
                DRAW_CAR: begin
                        //if car is done being drawn, plot is false
                        if(DoneDrawCar) plot = 1'b0;

                        //if car is not done being drawn
                        //draw car and plot are true
                        else begin
                                drawCar = 1'b1;
                                plot = 1'b1;
                        end
                end

                //when drawing over the car
                DRAW_OVER_CAR: begin

                        //if car isdone being drawn over, plot is false
                        if(DoneDrawErase) plot = 1'b0;
```

```
        //if car is not done being drawn over
        //drawovercar and plot are true
        else begin
                drawErase = 1'b1;
                plot = 1'b1;
        end
end

//when moving forward, move is true
MOVE_FORWARD: move = 1'b1;

//when moving left/right, move is true
MOVE_LEFT_RIGHT: move = 1'b1;

//when drawing explosions
DRAW_EXPLOSION: begin

        //if explosion is done being drawn, plot is false
        if(DoneDrawBoom) plot = 1'b0;

        //if explosion is not done being drawn
        //drawexplosion and plot are true
        else begin
                drawBoom = 1'b1;
                plot = 1'b1;
        end
end

//when drawing win screen, drawwinscreen and plot are true
DRAW_WIN_SCREEN: begin
```

```
                        drawWinScreen = 1'b1;
                        plot = 1'b1;
                    end
        endcase
    end // enable_signals


    // current_state registers
    always@(posedge Clock)
    begin: state_FFs

                    //when the game is being reset, set current state to set
reset signals
        if(!Resetn)
            current_state <= SET_RESET_SIGNALS;

                //when the game is not being reset, move on the state so
currentState = nextState
        else
            current_state <= next_state;
    end // state_FFS

endmodule
```

Appendix J: Instantiation of Datapath & Controlpath

```
module
projectT
op(

        //initalizes input for clock, resetn, SIMRESET----------------, start,
three direction turns
        input Clock, Resetn, simReset, Start, moveForward, moveRight, moveLeft,

        //initalizes output for the x position, y position, timer for the
score, colour display and plot
        output reg[8:0] xDisplay,
        output reg[7:0] yDisplay,
        output reg[7:0] secondsPassed,
        output[5:0] colourDisplay,
        output reg plotDisplay);

        //initialization of variables:

        //checking for reset,start of the race, drawing the background, car and
background with moving car
        wire setResetSignals, startRace, drawBG, drawCar, drawErase;

        //checking for whether car is moving, explosion needs to be drawn, win
screen and start screen display
        wire move, drawBoom, drawWinScreen, drawStartScreen;

        //checking for whether car, background, background with moving car is
done being drawn
        //checking if race is finished or if collision occured
        wire DoneDrawCar, DoneDrawBG, DoneDrawErase, FinishedRace, Collision;

        //checking if explosion, win screen, start screen has been drawn
```

```
        wire DoneDrawBoom, DoneDrawWinScreen, DoneDrawStartScreen;



//------------------------------------------------------------------------------
--------------------------
        wire[19:0] OneFrameCounter;



//------------------------------------------------------------------------------
--------------------------
        wire Enable1Frame;

        //Calls DelayCounter and feeds in a large count down number, clock,
simreset-----------------, and OneFrameCounter--------------------
        //Delay Counter module gives enough time for the user to see the
changes of the cars movement on the screen
        DelayCounter DC0(
                .countDownNum(20'd999_999), // SET TO 1 FOR SIMULATION PURPOSES
                .Clock(Clock),
                .simReset(simReset),
                .RDOut(OneFrameCounter));

        //Assigns EnableOneFrame to be true when OneFrameCounter is 0,
otherwise False
        //When true, it allows for the car to move foward and background to be
replaced
        assign EnableOneFrame = (OneFrameCounter == 20'd0) ? 1'b1 : 1'b0;

        //creates storage value for plotting the display
        wire plotWire;
```

```
        //calls the control function feeding it input for cock, reset,
EnableOneFrame and the states for all game situations
        control C0(
                .Clock(Clock),
                .Resetn(Resetn),
                .EnableOneFrame(EnableOneFrame),
                .start(Start),
                .forward(moveForward),
                .right(moveRight),
                .left(moveLeft),
                .DoneDrawBG(DoneDrawBG),
                .DoneDrawCar(DoneDrawCar),
                .DoneDrawErase(DoneDrawErase),
                .FinishedRace(FinishedRace),
                .Collision(Collision),
                .DoneDrawBoom(DoneDrawBoom),
                .setResetSignals(setResetSignals),
                .startRace(startRace),
                .drawBG(drawBG),
                .drawCar(drawCar),
                .drawErase(drawErase),
                .move(move),
                .drawBoom(drawBoom),
                .plot(plotWire),
                .DoneDrawStartScreen(DoneDrawStartScreen),
                .DoneDrawWinScreen(DoneDrawWinScreen),
                .drawStartScreen(drawStartScreen),
                .drawWinScreen(drawWinScreen));


        wire[8:0] xWire;
        wire[7:0] yWire;
```

```
datapath D0(
        .Clock(Clock),
        .Resetn(Resetn),
        .moveForward(moveForward),
        .moveRight(moveRight),
        .moveLeft(moveLeft),
        .setResetSignals(setResetSignals),
        .startRace(startRace),
        .drawBG(drawBG),
        .drawCar(drawCar),
        .drawErase(drawErase),
        .move(move),
        .drawBoom(drawBoom),
        .DoneDrawBG(DoneDrawBG),
        .DoneDrawCar(DoneDrawCar),
        .DoneDrawErase(DoneDrawErase),
        .FinishedRace(FinishedRace),
        .Collision(Collision),
        .DoneDrawBoom(DoneDrawBoom),
        .colourDisplay(colourDisplay),
        .xDisplay(xWire),
        .yDisplay(yWire),
        .DoneDrawStartScreen(DoneDrawStartScreen),
        .DoneDrawWinScreen(DoneDrawWinScreen),
        .drawStartScreen(drawStartScreen),
        .drawWinScreen(drawWinScreen));

always@(posedge Clock) begin
        xDisplay <= xWire;
```

```
            yDisplay <= yWire;
            plotDisplay <= plotWire;
        end

        wire[27:0] timer;
        wire EnableOneSecond;

        SecondsCounter S0(
            .Clock(Clock),
            .simReset(simReset),
            .countDownNum(28'd49_999_999),
            .RDOut(timer));

        assign EnableOneSecond = (timer == 28'd0) ? 1'b1 : 1'b0;

        always@(posedge Clock) begin
            if(EnableOneSecond) begin
                if(!FinishedRace) secondsPassed <= secondsPassed + 8'd1;
                else secondsPassed <= secondsPassed;
            end
            else begin
                if(!Resetn || !Start) secondsPassed <= 8'd0;
                else secondsPassed <= secondsPassed;
            end
        end

endmodule
```

Appendix K: Top Level Module

```
module
fill
                (

                        //initializes clock frequency
                        CLOCK_50,

                        //Initializes inputs for the key, switches, hex display + ps2
        inputs
                        KEY,
                        SW,
                        HEX0, HEX1,
                        PS2_CLK,
                        PS2_DAT,



                        // VGA related outputs
                        VGA_CLK,                                        //      VGA
        Clock
                        VGA_HS,                                         //      VGA
        H_SYNC
                        VGA_VS,                                         //      VGA
        V_SYNC
                        VGA_BLANK_N,                            //      VGA BLANK
                        VGA_SYNC_N,                                     //      VGA SYNC
                        VGA_R,                                          //      VGA
        Red[9:0]
                        VGA_G,                                          //      VGA
        Green[9:0]
                        VGA_B                                           //      VGA
        Blue[9:0]


                );
```

```
        //initializes clock frequency
        input                CLOCK_50;


        //Initializes inputs for the key, switches, hex display + ps2 inputs
        input    [3:0] KEY;
        input  [9:0] SW;
        output [0:6] HEX0, HEX1;


        // VGA related outputs
        output               VGA_CLK;                          //      VGA
Clock
        output               VGA_HS;                           //      VGA
H_SYNC
        output               VGA_VS;                           //      VGA
V_SYNC
        output               VGA_BLANK_N;              //      VGA BLANK
        output               VGA_SYNC_N;                       //      VGA SYNC
        output [7:0]  VGA_R;                           //      VGA Red[9:0]
        output [7:0]  VGA_G;                           //      VGA Green[9:0]
        output [7:0]  VGA_B;                           //      VGA Blue[9:0]


        //initializes variable for reset
        wire resetn;


        //assignes key[2] to reset the game
        assign resetn = KEY[3];


        //2 way stream for ps2
controller----------------------------------------------------------------------
-----
        inout                        PS2_CLK;
```

```
        inout                       PS2_DAT;
        wire                        signalStraight, signalRight, signalLeft;


        //Create the colour, x, y and writeEn wires that are inputs to the
controller.
        wire [5:0] colour;
        wire [8:0] x;
        wire [7:0] y;
        wire writeEn;
        wire[7:0] secondsPassed;


        // Create an Instance of a VGA controller - there can be only one!
        // Define the number of colours as well as the initial background
        // image file (.MIF) for the controller.
        vga_adapter VGA(
                        .resetn(resetn),
                        .clock(CLOCK_50),
                        .colour(colour),
                        .x(x),
                        .y(y),
                        .plot(writeEn),
                        /* Signals for the DAC to drive the monitor. */
                        .VGA_R(VGA_R),
                        .VGA_G(VGA_G),
                        .VGA_B(VGA_B),
                        .VGA_HS(VGA_HS),
                        .VGA_VS(VGA_VS),
                        .VGA_BLANK(VGA_BLANK_N),
                        .VGA_SYNC(VGA_SYNC_N),
```

```
        .VGA_CLK(VGA_CLK));

//loads the background with given resolution and file
defparam VGA.RESOLUTION = "320x240";
defparam VGA.MONOCHROME = "FALSE";
defparam VGA.BITS_PER_COLOUR_CHANNEL = 2;
defparam VGA.BACKGROUND_IMAGE = "track.mif";


projectTop P0(
        .Clock(CLOCK_50),
        .Resetn(resetn),
        .simReset(1'b0),
        .Start(SW[9]),
        .moveForward(signalStraight),
        .moveRight(signalRight),
        .moveLeft(signalLeft),
        .xDisplay(x),
        .yDisplay(y),
        .secondsPassed(secondsPassed),
        .colourDisplay(colour),
        .plotDisplay(writeEn));

PS2_Call ps2call(
        // Inputs
        CLOCK_50,
        SW[9],

        // Bidirectionals
        PS2_CLK,
```

```
                PS2_DAT,

                // Outputs
                signalStraight, signalRight, signalLeft);

        hex7seg hex0(.c(secondsPassed[3:0]), .led(HEX0));
        hex7seg hex1(.c(secondsPassed[7:4]), .led(HEX1));

endmodule
```

Appendix L: Hex 7 Seg

```
`timescale 1ns / 1ns //
`timescale
time_unit/time_precision
```

```
//code to display on hex0 and hex1
module hex7seg (input[3:0] c, output[0:6] led);


       assign led[0] =
(~c[3]&~c[2]&~c[1]&c[0])|(~c[3]&c[2]&~c[1]&~c[0])|

(c[3]&~c[2]&c[1]&c[0])|(c[3]&c[2]&~c[1]&c[0]);

       assign led[1] =
(~c[3]&c[2]&~c[1]&c[0])|(~c[3]&c[2]&c[1]&~c[0])|

(c[3]&~c[2]&c[1]&c[0])|(c[3]&c[2]&~c[1]&~c[0])|

(c[3]&c[2]&c[1]&~c[0])|(c[3]&c[2]&c[1]&c[0]);

       assign led[2] =
(~c[3]&~c[2]&c[1]&~c[0])|(c[3]&c[2]&~c[1]&~c[0])|

(c[3]&c[2]&c[1]&~c[0])|(c[3]&c[2]&c[1]&c[0]);

       assign led[3] =
(~c[3]&~c[2]&~c[1]&c[0])|(~c[3]&c[2]&~c[1]&~c[0])|

(~c[3]&c[2]&c[1]&c[0])|(c[3]&~c[2]&c[1]&~c[0])|

(c[3]&c[2]&c[1]&c[0]);
```

```
        assign led[4] =
(~c[3]&~c[2]&~c[1]&c[0])|(~c[3]&~c[2]&c[1]&c[0])|

(~c[3]&c[2]&~c[1]&~c[0])|(~c[3]&c[2]&~c[1]&c[0])|

(~c[3]&c[2]&c[1]&c[0])|(c[3]&~c[2]&~c[1]&c[0]);


        assign led[5] =
(~c[3]&~c[2]&~c[1]&c[0])|(~c[3]&~c[2]&c[1]&~c[0])|

(~c[3]&~c[2]&c[1]&c[0])|(~c[3]&c[2]&c[1]&c[0])|

(c[3]&c[2]&~c[1]&c[0]);


        assign led[6] =
(~c[3]&~c[2]&~c[1]&~c[0])|(~c[3]&~c[2]&~c[1]&c[0])|

(~c[3]&c[2]&c[1]&c[0])|(c[3]&c[2]&~c[1]&~c[0]);


endmodule
```

Appendix M: PS/2 Call

```
module
PS2_Call
(

        // Initializes a clock and reset input, alongside two bidirectional
inouts for the PS2
        input   CLOCK_50, input Resetn, inoutPS2_CLK, inout PS2_DAT,

        // Initializes three output wires for the three directions that the car
is allowed to move in the game
        output wire signalStraight, output wire signalRight, output wire
signalLeft
);

/*****************************************************************************
*                           Parameter Declarations                          *
*****************************************************************************/

// Instantiates with the inputs module that is later defined in this file
inputs detector(.CLOCK_50(CLOCK_50),
                                        .signalStraight(signalStraight),
                                        .signalLeft(signalLeft),
                                        .signalRight(signalRight),
                                        .ps2_key_pressed(ps2_key_pressed),
                                        .ps2_key_data(last_data_received),
                                        .Resetn(Resetn));

/*****************************************************************************
*                             Port Declarations                             *
```

```
**************************************************************************/

// Inputs

/*****************************************************************************
 *                 Internal Wires and Registers Declarations                 *
*****************************************************************************/

	// Internal Wires
	wire		[7:0]	ps2_key_data;
	wire				ps2_key_pressed;

	// Internal Registers
	reg			[7:0]	last_data_received;

	// State Machine Registers


/*****************************************************************************
 *                             Sequential Logic                              *
*****************************************************************************/

	always @(posedge CLOCK_50)
	begin
		// Only enters if the reset key is not pressed
		if(!Resetn)
```

```
                        // Stores
                        last_data_received <= 8'h00;
                else if(ps2_key_pressed == 1'b1)
                        last_data_received <= ps2_key_data;
        end



/*****************************************************************************
         *                              Internal Modules
*
*****************************************************************************/

        PS2_Controller PS2 (
                // Inputs
                .CLOCK_50(CLOCK_50),
                .reset(!Resetn),

                // Bidirectionals
                .PS2_CLK(PS2_CLK),
                .PS2_DAT(PS2_DAT),

                // Outputs
                .received_data        (ps2_key_data),
                .received_data_en     (ps2_key_pressed)
        );

endmodule
```

```
module inputs(CLOCK_50, signalStraight, signalLeft, signalRight,
ps2_key_pressed, ps2_key_data, Resetn);
// Inputs for the clock, reset, and the detection of which ps2 is pressed
input CLOCK_50,  ps2_key_pressed, Resetn;

input [7:0] ps2_key_data;
// Three directions that the car moves
output reg signalStraight, signalLeft, signalRight;

// The current state and next state for the finite state machine
reg[3:0] current_state, next_state;

       localparam
                           // "Make" code is sent when the key is pressed
down, and repeated periodically if the key is held down.
                           E0 = 4'd0,
                           // The "break" code is sent when the key is
released.
                           F0 = 4'd1,
                           // Initializes arbitrary values to the car
movements
                           WAIT = 4'd2,
                           LEFT = 4'd3,
                           RIGHT = 4'd4,
                           STRAIGHT = 4'd5,
                           LEFT_BREAK = 4'd6,
                           RIGHT_BREAK = 4'd7,
                           STRAIGHT_BREAK = 4'd8;

   // Next state logic
```

```
    always@(*)
    begin: state_table
                // The current state
                case (current_state)
                        WAIT: begin
                                // make
                                if (ps2_key_data == 8'hE0 && ps2_key_pressed)
next_state = E0;
                                // break
                                else if(ps2_key_data == 8'hF0 && ps2_key_pressed)
next_state = F0;
                                // goes back to wait, this is essentially a loop
to detect a keyboard click
                                else next_state = WAIT;
                        end

                        // begins if they key is pressed down or repeatedly held
                        E0: begin
                                // E0'75 is the hex for the up key
                                if(ps2_key_data == 8'h75) next_state = STRAIGHT;
                                // E0'6B is the hex for the left key
                                else if(ps2_key_data == 8'h6B) next_state = LEFT;
                                // E0'74 is the hex for the right key
                                else if(ps2_key_data == 8'h74) next_state = RIGHT;
                                // if they key is released, then move on to break
                                else if (ps2_key_data == 8'hF0) next_state = F0;
                                // detects a key press again
                                else next_state = WAIT;
                        end

                        // begins if the key is released
                        F0: begin
```

```
                                // E0'75 is the hex for the up key
                                if(ps2_key_data == 8'h75) next_state =
STRAIGHT_BREAK;
                                // E0'6B is the hex for the left key
                                else if(ps2_key_data == 8'h6B) next_state =
LEFT_BREAK;
                                // E0'74 is the hex for the right key
                                else if(ps2_key_data == 8'h74) next_state =
RIGHT_BREAK;
                                // detects a key press again
                                else next_state = WAIT;
                        end

                        // the movements detected will lead back to the wait
                        LEFT: next_state = WAIT;
                        RIGHT: next_state = WAIT;
                        STRAIGHT: next_state = WAIT;
                        LEFT_BREAK: next_state = WAIT;
                        RIGHT_BREAK: next_state = WAIT;
                        STRAIGHT_BREAK: next_state = WAIT;

                        // set default to wait as we want to know whether a key
is pressed or released
                        default: next_state = WAIT;
                endcase
        end


        always @(posedge CLOCK_50) begin
                // default false values for car movements
                if (!Resetn) begin
                        signalStraight <= 1'b0;
```

```
                signalLeft <= 1'b0;
                signalRight <= 1'b0;
            end

            // Sets signals to true if there is no break and false if there
is a break
            else if (current_state == STRAIGHT) signalStraight <= 1'b1;
            else if(current_state == LEFT) signalLeft <= 1'b1;
            else if(current_state == RIGHT) signalRight <= 1'b1;
            else if (current_state == STRAIGHT_BREAK) signalStraight <=
1'b0;
            else if (current_state == LEFT_BREAK) signalLeft <= 1'b0;
            else if (current_state == RIGHT_BREAK) signalRight <= 1'b0;


        end

        always@(posedge CLOCK_50)
        /*****************************************WHY
STATE_FFs*********************************/
   begin: state_FFs
            // default is to wait for user input
        if(!Resetn)
           current_state <= WAIT;
            // if reset is clicked, then the display will remain the same
and the current state will point to the next state
        else
            current_state <= next_state;
   end

endmodule
```