ECE385

Fall 2021

Final Project

Race Car Game on FPGA: "Asphalt"

Haozhe Si

haozhes3

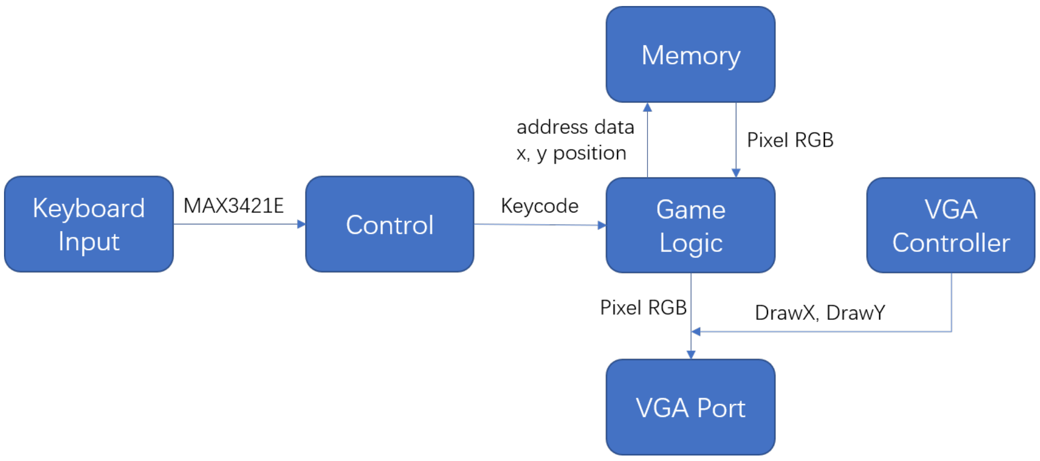
Section ABD

1. Introduction

In final project, we designed and implemented a race car game that involves avoiding obstacles and chasing target cars. Since the game has the same logic as the classical race car game *Asphalt*, we take that as the name of our game as well. This game is mainly implemented on the DE10 Lite FPGA board using System Verilog, while the keyboard control is implemented on software using NIOS-II. The game is controlled by the keyboard and the game graphic is generated by the VGA controller sending VGA signal directly to the VGA ports.

2. System Description

a) Overall Diagram for the System

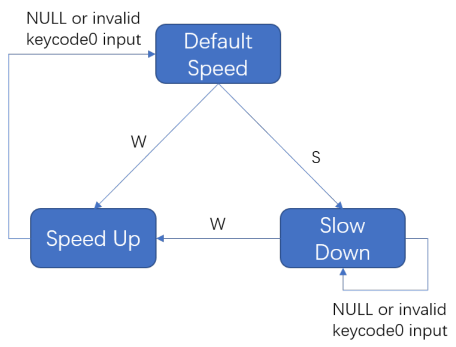


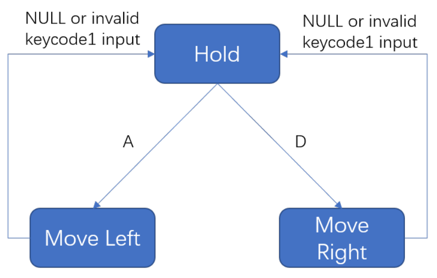
The major function of the NIOS-II CPU in this project is to communicate the control signal between the hardware and software. To support the communication between the NIOS-II and the MAX3421E, we created a low-level SPI interface. We also created several PIO to display data from software on the hardware and to read signal from the hardware to the software. After the hardware receives the control signal from the software, it will execute its control logic and fetching the memory data according to the game logic and output the RGB data according to the VGA controller. All the memories are located on the on-chip memory.

b) Input

The user can input the control via the keyboard. We take the code from Lab 6 for keypress detection and generate the keycodes. Since in the race car game, we want to control the direction and the speed of the race simultaneously, we modified the code in *main.c* and add an extra keycode port on the platform designer, so that it can generate two keycodes and send them to the hardware at the same time via two ports. On the hardware side, the System Verilog module *playerLogic* will recognize the input keycodes and generate the corresponding control signals for the game.

c) Player Logic


NULL or invalid keycode0 input
Default Speed
W
S
Speed Up
W
Slow Down
NULL or invalid keycode0 input


NULL or invalid keycode1 input
Hold
NULL or invalid keycode1 input
A
D
Move Left
Move Right

In play logic, horizontal movement and vertical movement will be treated separately. It is straight forward for the horizontal movement, where the car can move left or right according to the keystroke of A and D, whose keycodes are sent via the port keycode1, as long as the car is in the valid horizontal range. Things are more complicated for the vertical movement. Although the speed is controlled by the keystroke W and S directly, the game graphic generation is carefully designed. To increase the reality of the game and make the game easier to control, we will move the background and the cars in counter direction so that the cars can have a higher speed while not moving outside the screen. To be specific, the default speed for the car is 3 pixels per frame, the top speed will be 5, the speed when slowing down is 2 and the least speed is 1. However, in order to keep the car in screen all the time, we want the background moves as the car moves, and use the relative speed to generate the fact that the car is moving. Therefore, at the default speed, the car will not move while the background is moving backward at the speed of -3 pixels per frame. During speeding up, the ground holds the speed of -3, while the car has the speed of $speed - 3$. When the car reaches the top 1/3 of the screen, in order to keep the car in screen, the car can no longer move forward, but the ground will move at the speed of -5. Similarly, when slowing down, the car will have the speed of 2 and the ground has the speed of -3. Visually, the car will move backward. When the rear of the car hit the bottom of the screen, and S is still pressed, the speed will be 1. Meanwhile, as soon as the car reaches the bottom, the ground speed is decreased to the car speed, so that the car can hold still at the same place again. To sum up, the background speed is -3 when the players car does not hit the vertical bounds, otherwise, the background speed will equal the car speed but move backwards. The car location will be updated base on the relative speed between the ground speed and its actual speed.

d) Sprite and Memories

The sprite for the cars is saved in the on-chip memory. The sprite consists of the tile for 16 race cars in two rows with 8 cars on each row. The size of each race car is 47x67 pixels, with additional 4 pixels' horizontal interval. Thus, the total size of the sprite is 404x134 pixels. In order to save the memory space while preserving the color to the largest extend, we first quantize the sprite into

32 colors using the K-Means algorithm. Then, we can represent the colors in the sprite with 5-bit indices to the color palette. Therefore, we translate the quantized color into the color index using the python script. Due to the limitation of address bits, we will combine the indices of every four pixels into a bit-string with length 20. We further translate the bit-strings into a memory initialization file(.MIF). Therefore, the on-chip memory will have 16,384 20-bit words, while 13,534 of the words used for the sprite. Notice that a specific color is assigned for the transparent part in the tile, and the hardware will ignore that pixel if read that index. We implemented a on-chip memory module using the Megafunction and load the MIF file as the initialization of the OCM. Later, we can access the color index for the pixels using the index of the car we want to paint and the location of the pixel on the tile, and output the pixel value from the color palette.

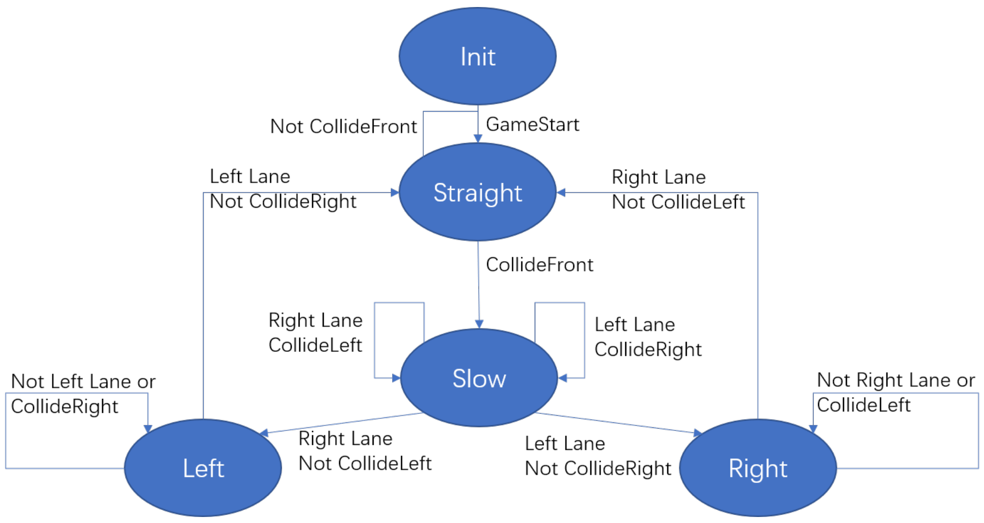
e) Car Stream Generation and Collision Detection

We designed four lanes for the race track of the game, two four each direction. The two lanes also have different car speeds, imitating lane conditions of the real highway. To assign the car stream, we define a map for the cars at each lane. Given a location on the Y axis of the screen and the lane, we can check the dictionary to see if there is a car at this slot. Since car on different lanes have different speed and moving direction, we need to control the movement of the car streams on different lanes separately. Therefore, instead of using Y axis position as the index to the map, we use the Y distance, which this the accumulation of the car speed for each lane at each frame clock. The Y distance offsets the car stream map so that the car streams can move at different speed and direction on each lane.

With the car stream generated, we also implemented the collision detection. We define the horizontal and vertical collision box for the player's car and the AI car. The collision detector will check if these cars collide with the car stream. If collided, it can also tell the car is collided at which direction. The collide detection is especially useful in designing the AI for the target car. The implementation detail will be discussed in the module description section.

f) AI Car Logic

The AI car will only run on the right two lanes of the road. It is controlled by the current lane its on and the collision signal calculated base on its current location. The AI car runs at the speed of 4 pixels per frame by default, which is faster than the car speed on both lanes. Therefore, it is inevitable to collide with the front car. We define the collision box for the AI car to be 20 pixels, so that the AI car can have enough time to make response to the collision. If the AI car is collided with the front car, it will first slow down to the front car speed. Then, depending on its current lane, AI car will turn left or turn right to avoid the front car. If the collision signal is also set for the other lane, the AI car will just wait until the obstacle car pass by. If no more obstacle on the other lane, the AI car will turn until it is completely on the new lane. The naïve autonomous car AI can avoid the obstacle cars smoothly.



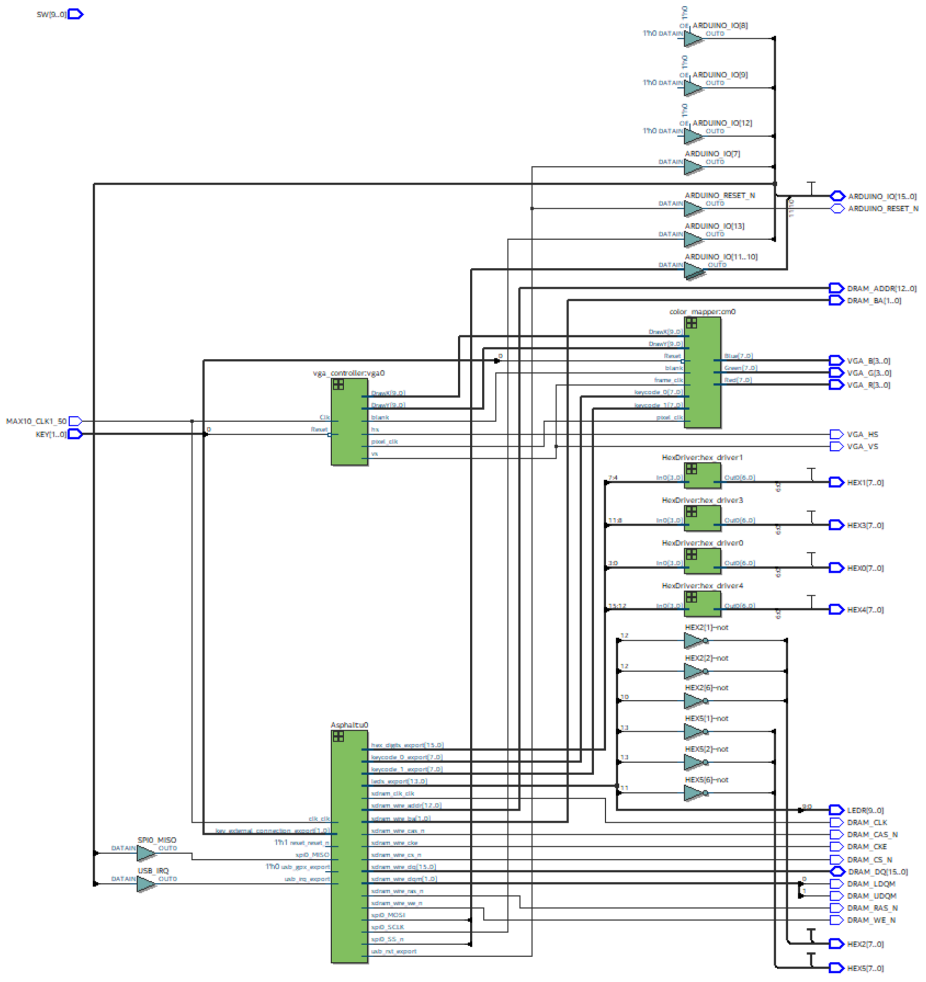
g) Game Goal

The goal of the game is that the player needs to control the car to chase the AI target car. If the player car catches the AI car, the player wins. If the distance between the player can and the AI car exceeds a threshold, or the player keeps colliding with the obstacle for a long time, the player loses.

h) VGA Display

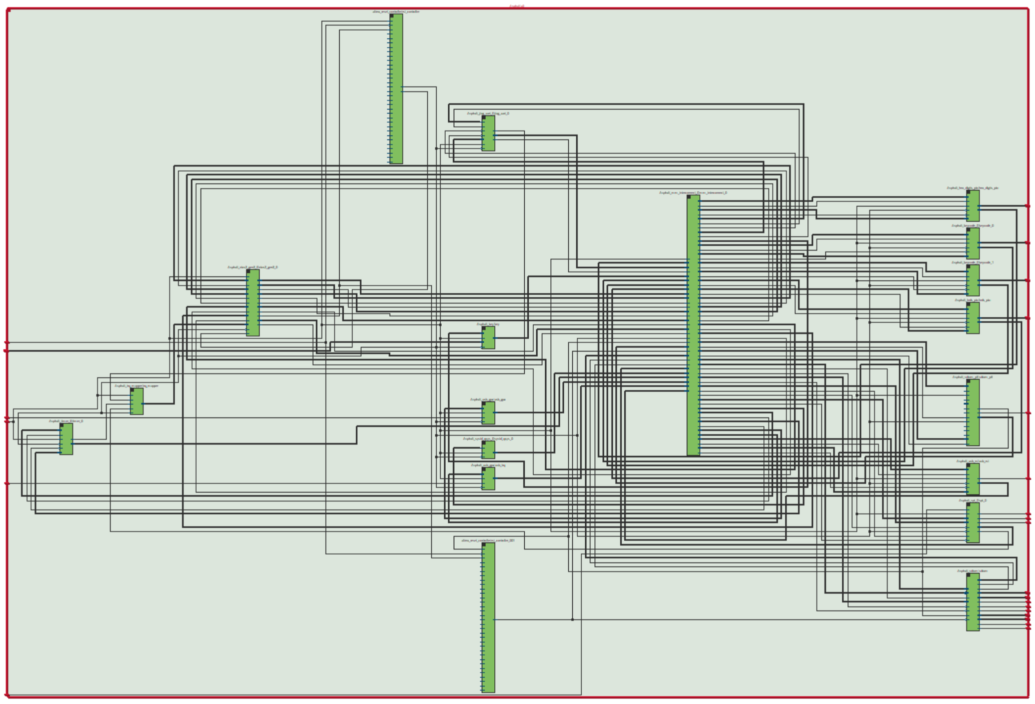
The color mapper decides which color to be output to the VGA port. Inside the color mapper, we instantiated the logic controllers and the corresponding game component plotter. Each plotter takes in the *DrawX* and *DrawY* signals generated by the VGA Controller, and output the RGB signal for that pixel location. In addition to the RGB signals, the plotter will output signal to tell if the current location is a valid position for their object. If is not, the color mapper will ignore this output RGB signal. The color mapper will then output the RGB signals according to the preset priority sequence of every object. As the VGA Controller module traverses every pixel of the monitor, the color mapper will generate correct RGB values.

3. Top Level Block Diagram



4. Module Description

Module: Asphalt_gametop.sv
Inputs: [1:0] KEY, [7:0] SW, MAX10_CLK1_50
Outputs: [7:0] LEDR,
[7:0] HEX0, [7:0] HEX1, [7:0] HEX2, [7:0] HEX3, [7:0] HEX4, [7:0] HEX5, VGA_HS, VGA_VS, [3:0] VGA_R, [3:0] VGA_G, [3:0] VGA_B,
[12:0] DRAM_ADDR, DRAM_CKE, [1:0] DRAM_BA, DRAM_CLK, DRAM_CAS_N, DRAM_CS_N, DRAM_LDQM, DRAM_UDQM, DRAM_RAS_N, DRAM_WE_N
Inout: [15:0] DRAM_DQ, [15:0] ARDUINO_IO, ARDUINO_RESET_N
Description: This module instantiated the module for NIOS-II system, the VGA controller, and the color mapper.
Purpose: This is the top-level module for interfacing with the NIOS-II system module. It connects the IOs and DRAM on the MAX10 board to the NIOS-II system module.

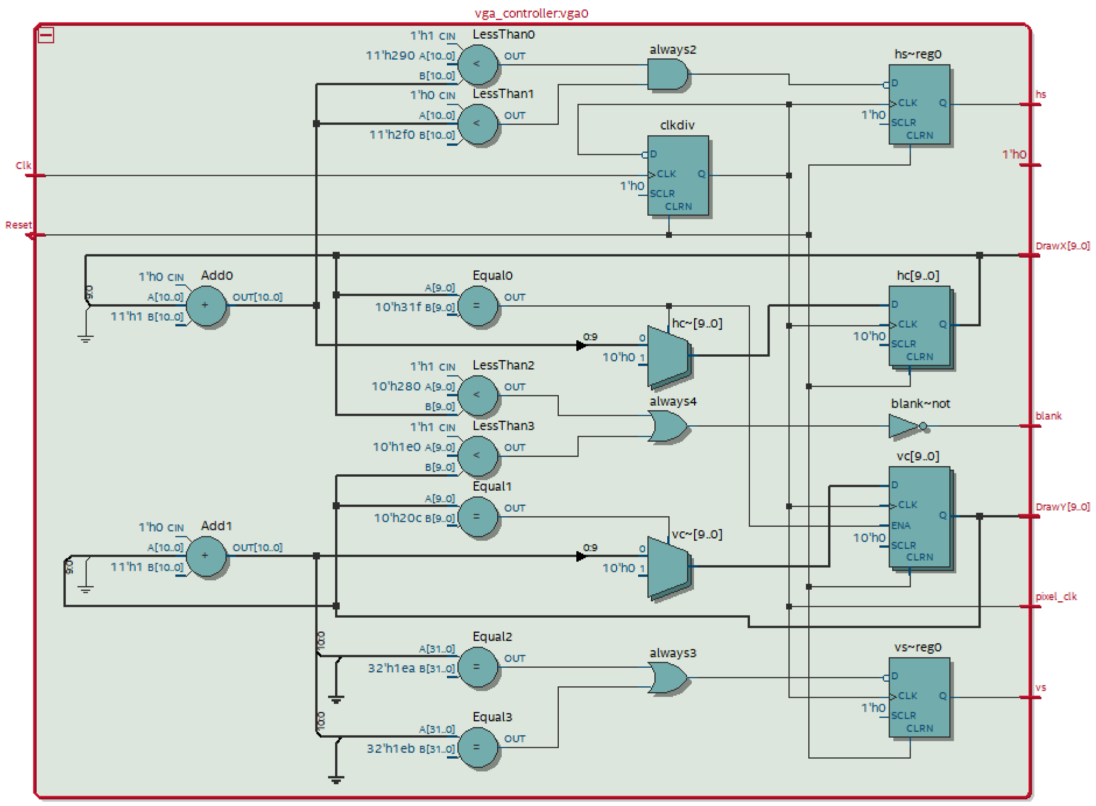


Module: Asphalt.v

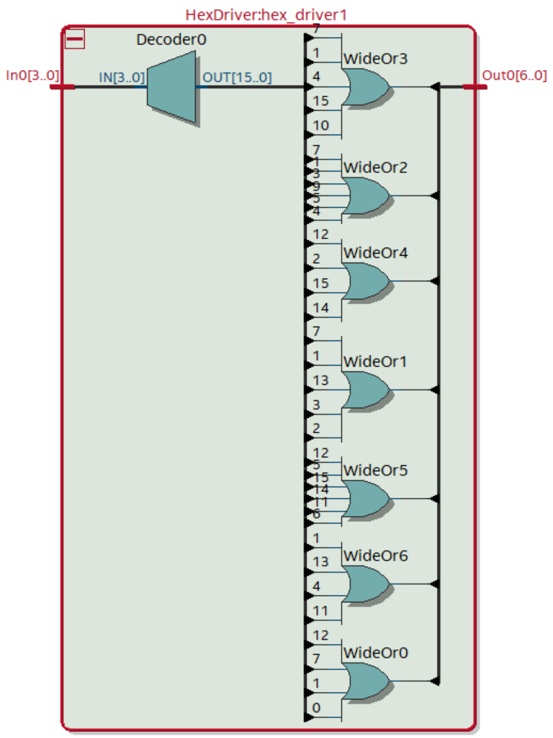
Components: clk_0, nios2_gen2_0, leds_pio, sdram, sdram_pll, sysid_qsys_0, keycode_0, keycode_1, hex_digits_pio, spi_0, jtag_uart_0, timer_0, usb_irq, usb_gpx, usb_rst, key

Description: This module instantiated the components used in NIOS-II system. Among all the components, the PIO blocks are the *leds_pio* and the *hex_digits_pio* block, which are used for displaying the results of operation; the *keycode_0* and *keycode_1* block, which outputs the keycode of the key stroke to the FPGA, the *key* block, which reads the push button signal to the NIOS-II system, and the *usb_irq, usb_gpx, usb_rst* signals, which communicate with the USB chip.

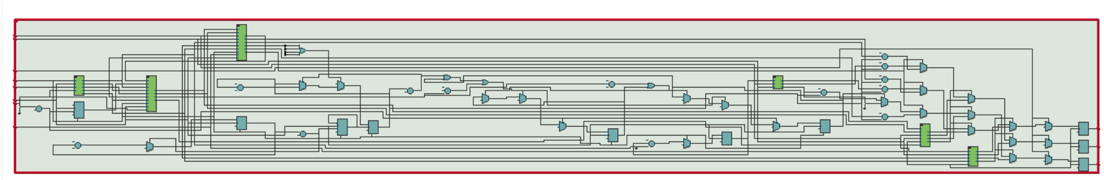
Purpose: This is the System Verilog code for the SOC we defined in the QsYs. The components are connected according to the code we have here.



Module: VGA_controller
Inputs: Clk, Reset
Outputs: hs, vs, pixel_clk, sync, blank, [9:0] DrawX, [9:0] DrawY
Description: This module will first divide the *Clk* speed in half and create the *pixel_clk*. Then module is then synchronized with the *pixel_clk*. The VGA_controller module will scan the pixel array of the monitor. Every time the it finishing scanning the length of a row, it will set the *hs* signal, and every time it finishing all pixels, it will set the *vs* signal. Since the actual scanning area is larger than the screen area, when the controller scans the area outside the monitor, it will set the *blank* signal. The module will also output the coordination of the pixels it is scanning at the current clock cycle.
Purpose: The purpose of this module is to decide the location of the pixel it is scanning. It also outputs the synchronization signals of the VGA hardware. This information will be later used to determine the color of the pixels.

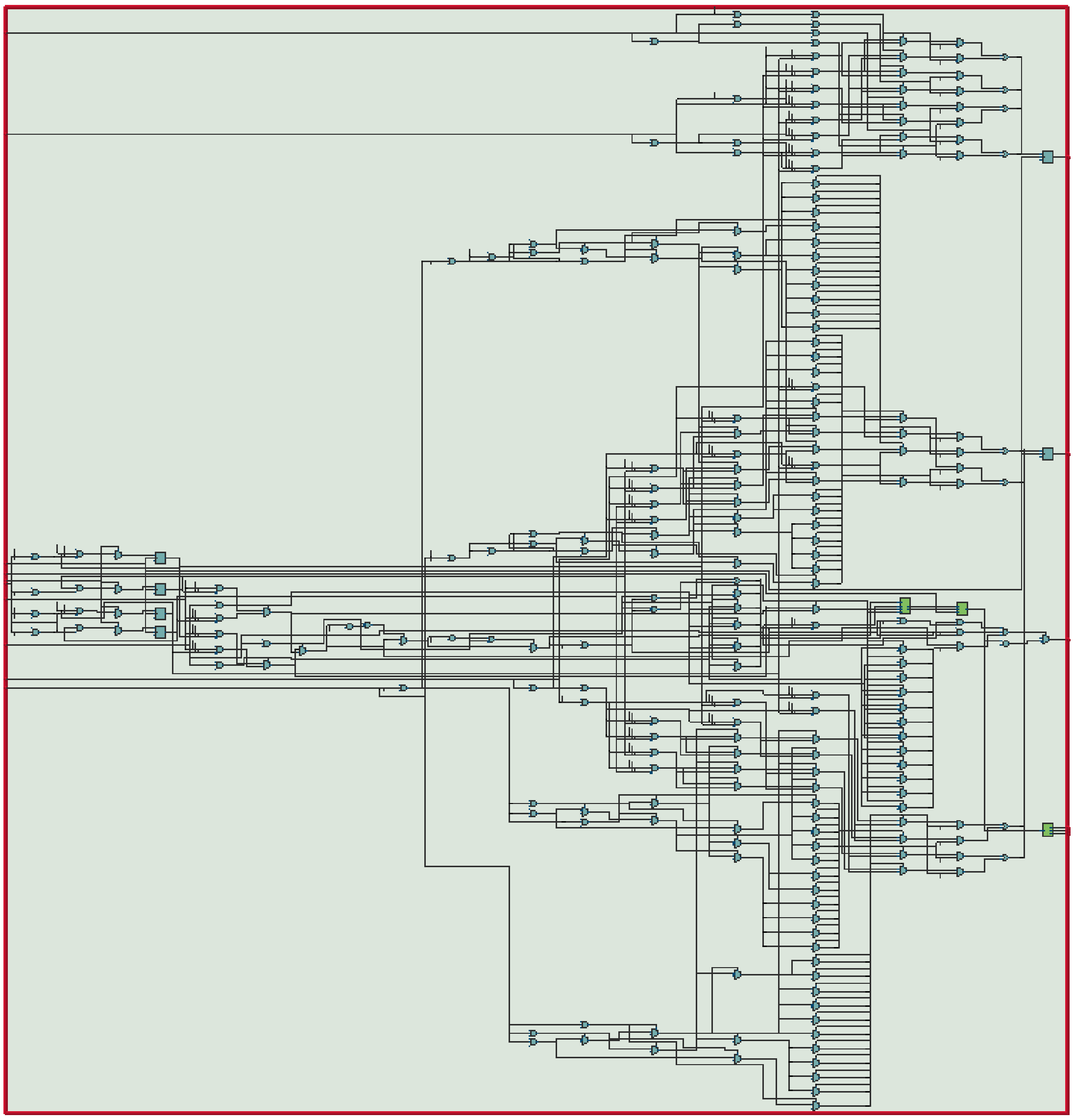


Module: HexDriver
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: The module will output the code for seven-segment display depending on the input 4-bit data.
Purpose: This module is used to translate the binary data into the code for hex display on the board.



Module: Color_Mapper
Inputs: [9:0] DrawX, [9:0] DrawY, [7:0] keycode_0, [7:0] keycode_1, blank, Reset, frame_clk, pixel_clk
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue
Description: The Color Mapper module instantiated the game logic modules and the plotter modules. The game logic modules will output the location of the objects and the plotter modules will output the corresponding RGB value for the objects given their location and the current *DrawX* and *DrawY*. The color mapper module aggregates this information and output the RGB signal for the current *DrawX* and *DrawY* base on the outputs from the plotters and a preset priority list. In addition, the color mapper will also handle the game state and display the game state by manipulating the color style of the game graphic.

Purpose: This module performs aggregate the control signals for the game graphic and output the RGB signal for the pixel depending on the control signals.



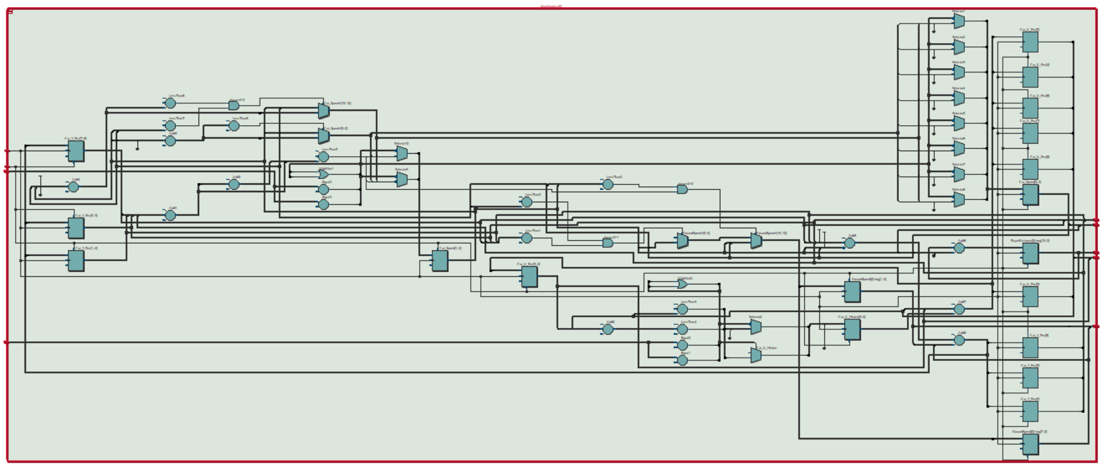
Module: plotObstacles

Inputs: [9:0] DrawX, [9:0] DrawY, [9:0] PlayerX, [9:0] PlayerY, [9:0] AIX, [9:0] AIY, [9:0] GroundSpeed, clk, Reset, frame_clk

Outputs: [7:0] Red, [7:0] Green, [7:0] Blue, [3:0] PlayerCollide, [3:0] AICollide, isObs

Description: This module will generate the obstacle on the lane, i.e., the car stream. As we discussed before, we first defined an obstacle map for each lane. By keeping track of the distance each lane travelled, we can track which part of the map should be print on the screen and what is the offset. Similarly, we can use the map to check collision. We defined the horizontal and vertical collision box for the player and the AI cars. Given the four vertices of the car, we add the width of the collision box vertically or horizontally to the vertices and consequently define eight key points for the car, with two points for an edge. Then, we will decide the lane the key points are on. With the lane information and the Y distance, we can check map and find if the places the key points are at are occupied by the obstacle cars. If one of the key points are occupied on an edge, the collision signal will be set on that edge.

Purpose: This module will plot the obstacle car given the *DrawX* and the *DrawY* signals at each pixel clock, and check the collision for the player car and the AI car with the obstacles at each frame given their locations.



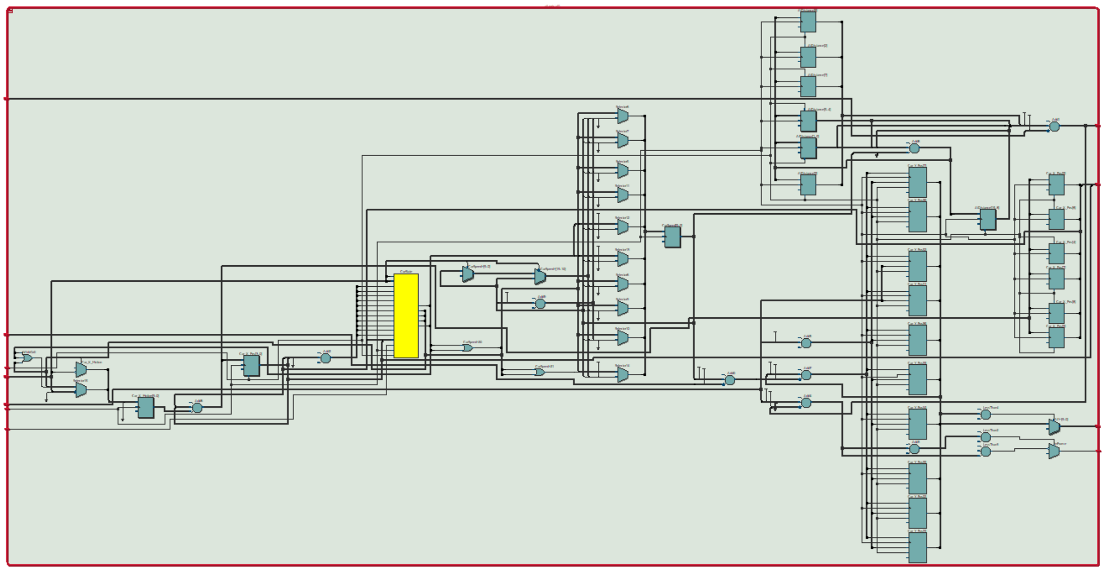
Module: playerLogic

Inputs: [7:0] keycode_0, [7:0] keycode_1, Reset, frame_clk

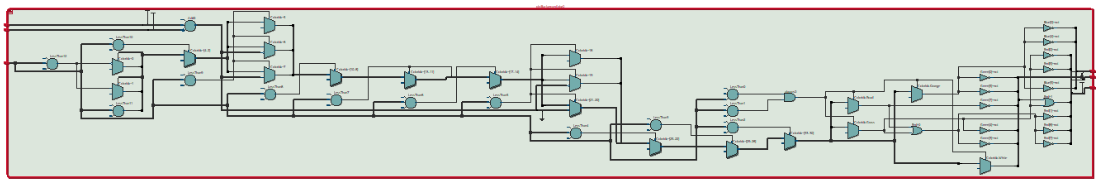
Outputs: [15:0] PlayerDistanc, [9:0] PlayerX, [9:0] PlayerY, [9:0] PlayerSpeed, [9:0] GroundSpeed,

Description: This module will take in the key board control and decide the movement of the player car. When *Reset* is high, the player position will be reset to the initial place. Then, at each frame clock, the module will update the location of the car on the screen base on the current car speed, ground speed and control signals as discussed before. The net Y axis movement value at a frame will be the different of the player speed and the ground speed. The module output the player location after the update, as well as the current player speed and the ground speed. The *PlayerDistance* is the distance the player travelled. This signal is necessary for checking if the player has caught the AI car.

Purpose: This module will decide the player location, the player speed, and the ground speed. It will also keep track of the distance the player travelled. The player location is used for plotting the player on screen. The ground speed is used as the benchmark for the other cars to calculate the relative speed. The player distance is used for game logic control.

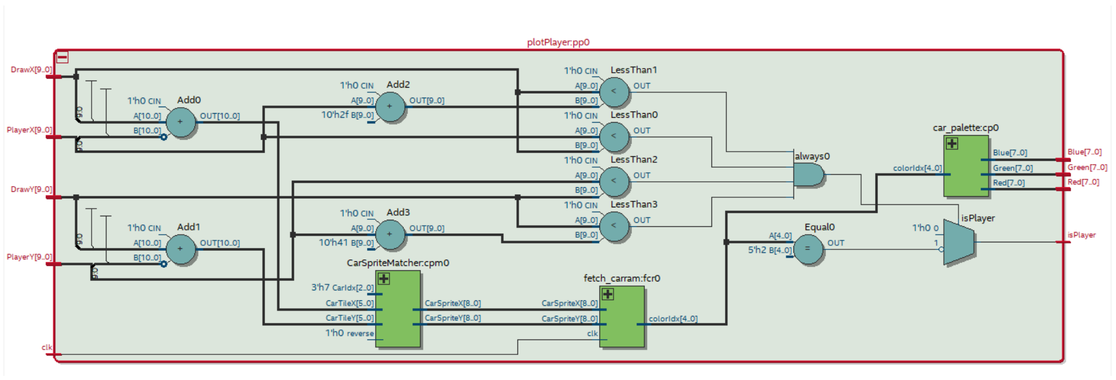


Module: aiLogic
Inputs: [15:0] PlayerDistance, [9:0] PlayerX, [9:0] PlayerY, [9:0] GroundSpeed, [9:0] PlayerSpeed, [7:0] keycode_1, [3:0] AICollide, frame_clk, Reset, GameStart,
Outputs: [15:0] TarDistance, [9:0] AIX, [9:0] AIY, inFrame
Description: This module will take in the collision signal and decide the movement of the AI car. When *Reset* is high, the AI car position will be reset to the initial place. Then, at each frame clock, the module will update the location of the car on the screen base on the current car location, ground speed and collision information signals as discussed before. The net Y axis movement value at a frame will be the different of the AI car speed and the ground speed. The module output the AI car location after the update. The module will also keep track of the distance the AI car travelled. The difference between distance travelled of the AI car and the player car, *TarDistance*, is the distance between them. If the *PlayerY* signal minus this distance is smaller than zero, that means the AI car is off screen. Then, the *inFrame* signal will be set to zero and the plot AI module will ignore the AI car. The *TarDistance* signal is necessary for checking if the player has caught the AI car as well.
Purpose: This module will decide the AI car location. It will also keep track of the distance the AI car travelled and calculate the distance between the player car and the AI car. Base on the distance and the player's Y location, the module will further decide if the AI is in the screen.

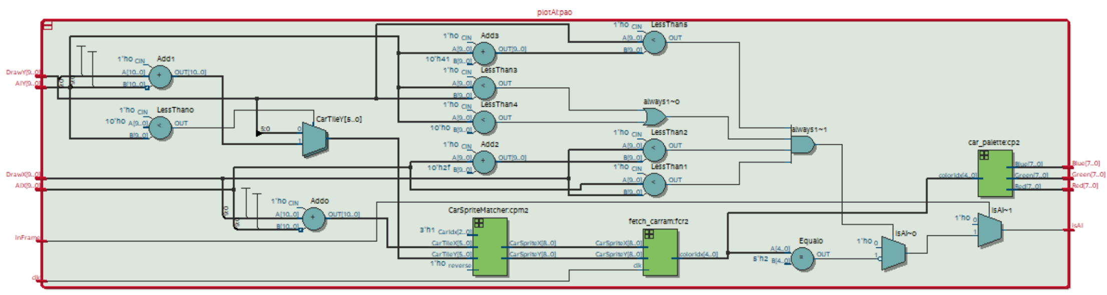


Module: plotBackground
Inputs: [9:0] DrawX, [9:0] DrawY, [10:0] Distance
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue
Description: This module will decide the region of the background base on *DrawX*

and output the corresponding color given the region. Notice that the dotted lines between each lane on the same direction needs to move at the speed of ground speed. This can be offset by the *DrawY* signal and the *Distance*.
Purpose: The module is used to plot the moving background as the VGA control signal sweep the screen.



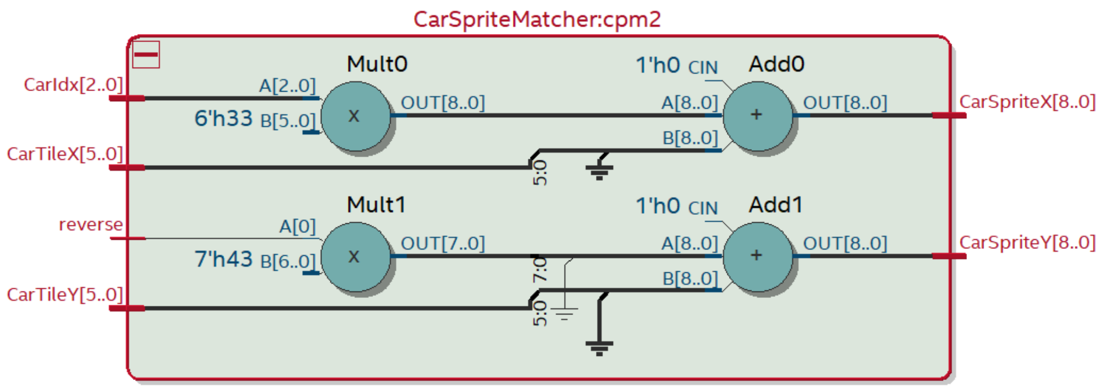
Module: plotPlayer
Inputs: [9:0] DrawX, [9:0] DrawY, [9:0] PlayerX, [9:0] PlayerY, clk
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue, isPlayer
Description: This module will plot the play given its location. If the *DrawX* and the *DrawY* signals are not in the region of the car tile, the *isPlayer* signal will be set to low and the color mapper will ignore the RGB output. If the pixel is at the valid pixel for the car tile, this module will calculate the pixel's location on the tile. By specifying the index of the car we want to plot, the *CarIdxMatcher* module will calculate the memory address for the pixel and the *fetch_carram* module will fetch the corresponding color index for the pixel. The *car_palette* module will output the RGB signal for the pixel.
Purpose: The module is used to plot the player car given its location.



Module: plotAI
Inputs: [9:0] DrawX, [9:0] DrawY, [9:0] AIX, [9:0] AIY, clk, inFrame
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue, isAI
Description: This module will plot the play given its location. If the *inFrame* signal is low, the *isAI* signal will be set to low and the color mapper will ignore the RGB output. If the *DrawX* and the *DrawY* signals are not in the region of the car tile, the *isAI* signal will be set to low as well. If the pixel is at the valid pixel for the car tile, this module will calculate the pixel's location on the tile. By specifying the index

of the car we want to plot, the *CarSpriteMatcher* module will calculate the memory address for the pixel and the *fetch_carram* module will fetch the corresponding color index for the pixel. The *car_palette* module will output the RGB signal for the pixel.

Purpose: The module is used to plot the AI car given its location.



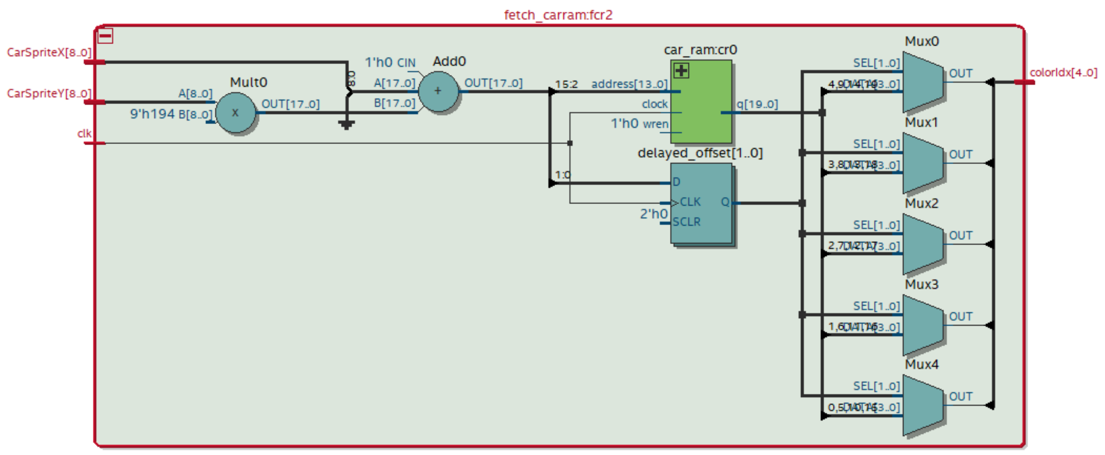
Module: CarSpriteMatcher

Inputs: [5:0] CarTileX, [5:0] CarTileY, [2:0] CarIdx, reverse

Outputs: [8:0] CarSpriteX, [8:0] CarSpriteY

Description: This module will calculate the pixel location on the sprite given the pixel location on the tile and the car index.

Purpose: The module is used for finding the pixel location on the sprite given the tile location and the target car we want to plot.



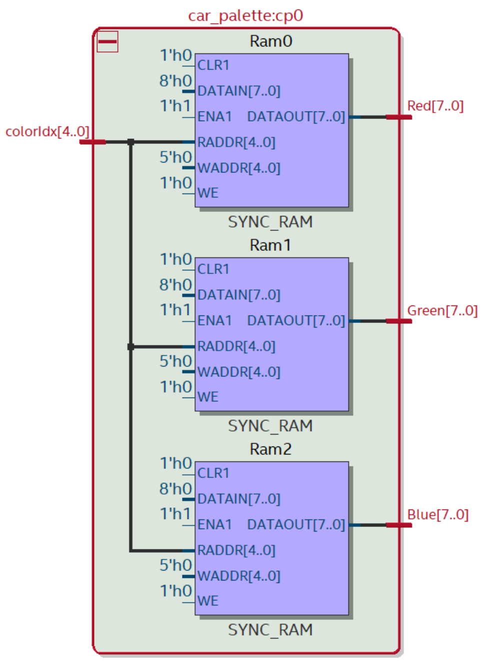
Module: fetch_carram

Inputs: [8:0] CarSpriteX, [8:0] CarSpriteY, clk

Outputs: [4:0] colorIdx

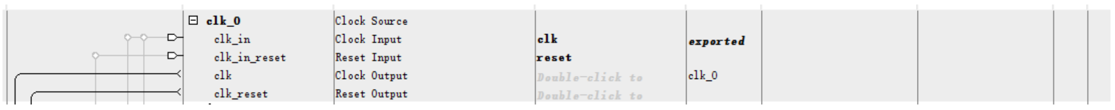
Description: This module will fetch out the color index given the pixel location on the sprite by calculating its address and offset on the OCM.

Purpose: The module is used for finding the color index for the input sprite location.



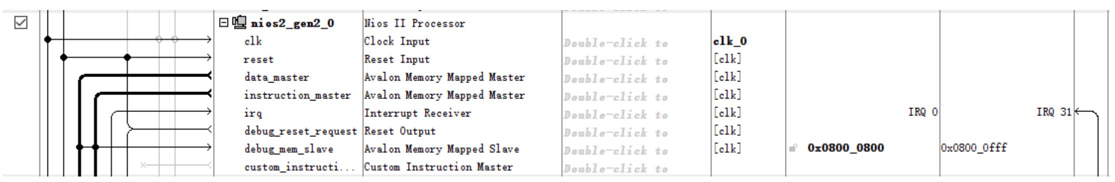
Module: car_palette
Inputs: [4:0] colorIdx
Outputs: [7:0] Red, [7:0] Green, [7:0] Blue
Description: This module is a dictionary and will give out the RGB value given the color index.
Purpose: The module is used for finding the color value given the color index.

5. System Level Block Diagram



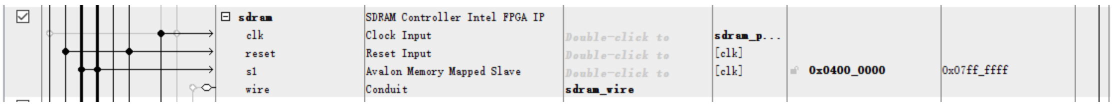
clk_0	Clock Source		
clk_in	Clock Input	clk	exported
clk_in_reset	Reset Input	reset	
clk	Clock Output	Double-click to	clk_0
clk_reset	Reset Output	Double-click to	

Module: clk_0
Exports: clk, reset
Description: This module is the clock source for the NIOS-II system.



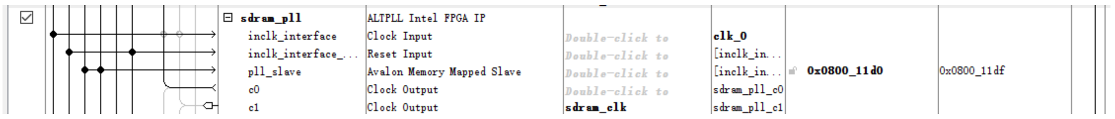
nios2_gen2_0	Nios II Processor					
clk	Clock Input	Double-click to	clk_0			
reset	Reset Input	Double-click to	[clk]			
data_master	Avalon Memory Mapped Master	Double-click to	[clk]			
instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]			
irq	Interrupt Receiver	Double-click to	[clk]	IRQ 0		IRQ 31
debug_reset_request	Reset Output	Double-click to	[clk]			
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_0800	0x0800_0fff	
custom_instructi...	Custom Instruction Master	Double-click to				

Module: nios2_gen2_0
Exports:
Description: The economy version processor of the NIOS-II system. It handles all the logical operations and computations the programs might use.



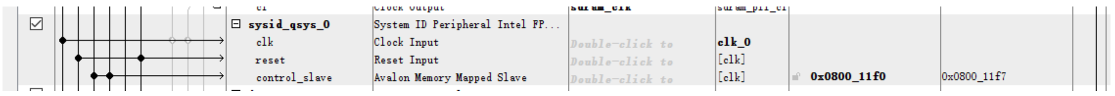
⊟ sdram	SDRAM Controller Intel FPGA IP				
clk	Clock Input	Double-click to	sdram_p...		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0400_0000	0x07ff_ffff
wire	Conduit	sdram_wire			

Module: sdram
Exports: sdram_wire
Description: The SDRAM controller of the NIOS-II system. It specifies the size the of the SDRAM and the SDRAM IP module will be the driver for memory, performing operations like refreshing every time we do memory operations.



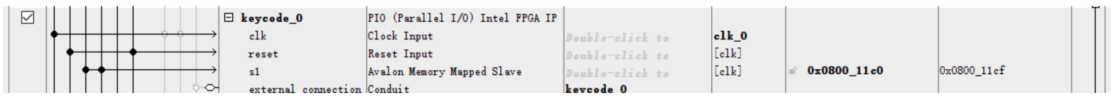
⊟ sdram_pll	ALTPLL Intel FPGA IP				
inclk_interface	Clock Input	Double-click to	clk_0		
inclk_interface_...	Reset Input	Double-click to	[inclk_in...		
pll_slave	Avalon Memory Mapped Slave	Double-click to	[inclk_in...	0x0800_11d0	0x0800_11df
c0	Clock Output	Double-click to	sdram_pll_c0		
c1	Clock Output	sdram_clk	sdram_pll_c1		

Module: sdram_pll
Exports: sdram_clk
Description: Pll means "Phase Lock Loop". This module will generate a second clock signal with a phase shift. This block is necessary to prevent skew operation.



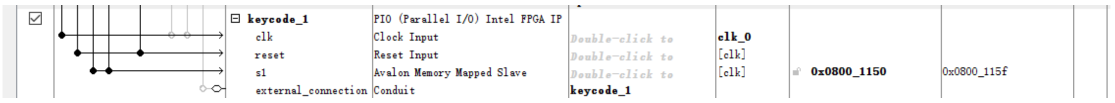
⊟ sysid_qsys_0	System ID Peripheral Intel FP...				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
control_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_11f0	0x0800_11f7

Module: sysid_qsys_0
Exports:
Description: The system ID checker is used to ensure the compatibility between the hardware and software we defined. This module will provide a serial number, and the software loader will check this number at the beginning of the program. This prevents us from loading software with incompatible NIOS-II configuration to FPGA.



⊟ keycode_0	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_11c0	0x0800_11cf
external_connection	Conduit	keycode_0			

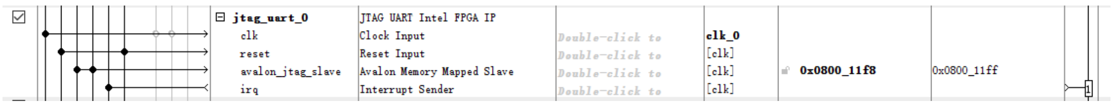
Module: keycode_0
Exports: keycode_0
Description: This is a PIO block that reads the keycode of the key stroke from the keyboard. The exported keycode will be connected to the physical hex digit display pins on the FPGA board so that we can display the presses keycode on the hex digit displays. This signal will also be used to control the horizontal movement of the Player Car.



⊟ keycode_1	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1150	0x0800_115f
external_connection	Conduit	keycode_1			

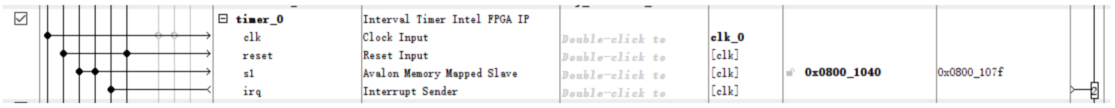
Module: keycode_1
Exports: keycode_1
Description: This is a PIO block that reads the a second keycode of the key stroke from

the keyboard. This signal will also be used to control the speed of the Player Car.



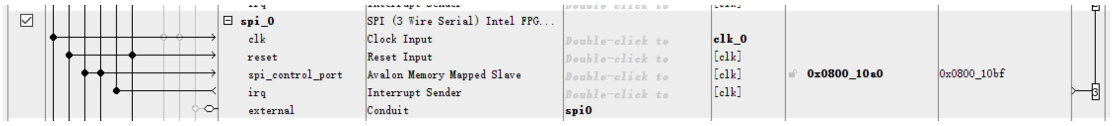
jtag_uart_0	JTAG UART Intel FPGA IP					
clk	Clock Input	Double-click to	clk_0			
reset	Reset Input	Double-click to	[clk]			
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_11f8	0x0800_11ff	
irq	Interrupt Sender	Double-click to	[clk]			1

Module: jtag_uart_0
IRQ: IRQ1
Description: This JTAG_UART block is used for debug. It has an interrupt connection with the processor. With this block, we can use "printf" in Ellipse.



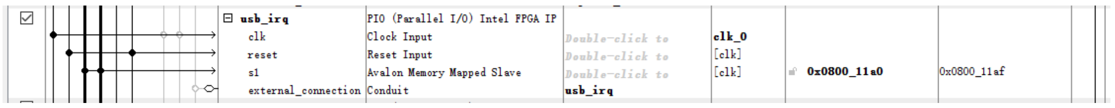
timer_0	Interval Timer Intel FPGA IP					
clk	Clock Input	Double-click to	clk_0			
reset	Reset Input	Double-click to	[clk]			
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1040	0x0800_107f	
irq	Interrupt Sender	Double-click to	[clk]			2

Module: timer_0
IRQ: IRQ2
Description: This is a timer block that counts on millisecond intervals. It has an interrupt connection with the processor. The block is necessary in the USB driver code in order to keep track of the various time-outs that USB requires.



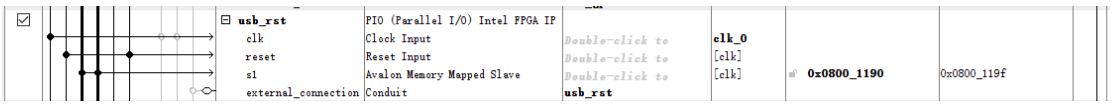
spi_0	SPI (3 Wire Serial) Intel FPG...					
clk	Clock Input	Double-click to	clk_0			
reset	Reset Input	Double-click to	[clk]			
spi_control_port	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_10a0	0x0800_10bf	
irq	Interrupt Sender	Double-click to	[clk]			3
external	Conduit	spi0				

Module: spi_0
IRQ: IRQ3
Description: This is a SPI block that communicate with the USB controller with four buses: SS, MOSI, MISO, SCLK. The functionality of the SPI block is discussed above. The four signals of SPI will connect to the corresponding ports of the MAX3421E USB Peripheral/Host Controller. It will also have an interruption with the processor.



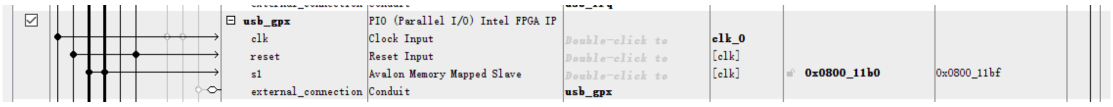
usb_irq	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_11a0	0x0800_11af
external_connection	Conduit	usb_irq			

Module: usb_irq
Export: usb_irq
Description: This is a PIO block which connects to the interrupt output of the MAX3421E USB Peripheral/Host Controller as required by datasheet.



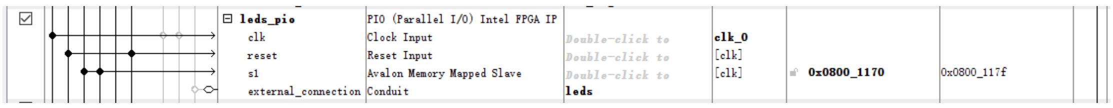
usb_rst	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1190	0x0800_119f
external_connection	Conduit	usb_rst			

Module: usb_rst
Export: usb_irst
Description: This is a PIO block which connects to the device reset of the MAX3421E USB Peripheral/Host Controller as required by datasheet.



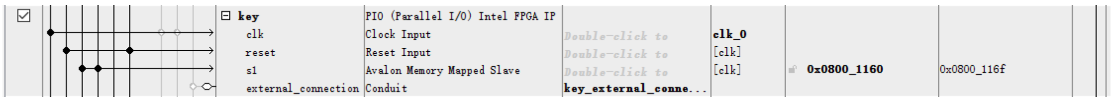
⊟ usb_gpx	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_11b0	0x0800_11bf
external_connection	Conduit	usb_gpx			

Module: usb_gpx
Export: usb_gpx
Description: This is a PIO block which connects to the General-Purpose Multiplexed Push-Pull output of the MAX3421E USB Peripheral/Host Controller as required by datasheet.



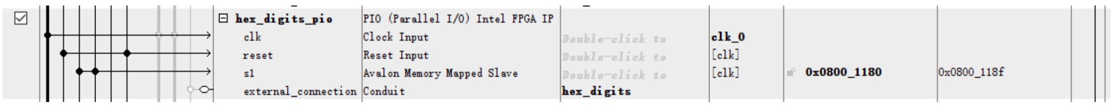
⊟ leds_pio	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1170	0x0800_117f
external_connection	Conduit	leds			

Module: led_pio
Exports: leds
Description: This is a PIO block that handles the led output. The exported leds will be connected to the physical led pins on the FPGA board so that we can control the led through software using NIOS-II system.



⊟ key	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1160	0x0800_116f
external_connection	Conduit	key_external_conne...			

Module: key
Exports: key_external_connections
Description: This is a PIO block that handles the push button input. The exported key_external_connections will be connected to the physical push button pins on the FPGA board so that we can read the push button inputs through software using NIOS-II system.

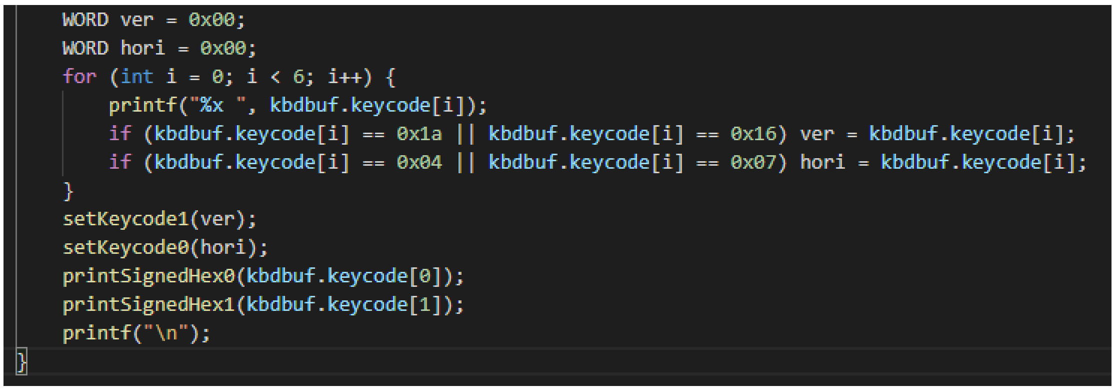


⊟ hex_digits_pio	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_1180	0x0800_118f
external_connection	Conduit	hex_digits			

Module: hex_digits_pio
Exports: hex_digits
Description: This is a PIO block that handles the hex digits display output. The exported hex_digits will be connected to the physical hex digits display on the FPGA board so that we can control the display through software using NIOS-II system.

6. Software Components

In this project, we used the software to read the keystrokes from the keyboard through MAX3421E chip and control the hex digit display. The software codes are basically the same as what we used in Lab 6, except that we add another keycode port. The modified *main.c* function will explicitly check for the keycode of W, A, S and D. The main function will set the horizontal movement keys (A and D) to keycode0 and the vertical movement keys (W and S) to keycode 1. All the reset part of the code are remained exactly the same.



```
    WORD ver = 0x00;
    WORD hori = 0x00;
    for (int i = 0; i < 6; i++) {
        printf("%x ", kbdbuf.keycode[i]);
        if (kbdbuf.keycode[i] == 0x1a || kbdbuf.keycode[i] == 0x16) ver = kbdbuf.keycode[i];
        if (kbdbuf.keycode[i] == 0x04 || kbdbuf.keycode[i] == 0x07) hori = kbdbuf.keycode[i];
    }
    setKeycode1(ver);
    setKeycode0(hori);
    printSignedHex0(kbdbuf.keycode[0]);
    printSignedHex1(kbdbuf.keycode[1]);
    printf("\n");
}
```

7. Design Analysis

LUT	5020
DSP	0
Memory (BRAM)	994304
Flip-Flop	643
Frequency	74.6MHz
Static Power	94.76mW
Dynamic Power	107.40mW
Total Power	226.33mW

8. Conclusion

In the final project, the pixel version of Asphalt is implemented. We made use of the knowledge we learned in the past semester to build this project, including designing of finite state machine for control, the implementing the peripherals in platform designer, generating the VGA signals, and initializing the on-chip memory. In the end, our project works well. However, due to the limitation of time, we did not add much more extra functionalities. The project is hard then we initially expected, especially for the hard AI car control. It cost us much time to figure out the most reasonable approach to perform the collision detection. We implemented the logic control part of project purely on the hardware, it is challenging for us, but also satiating when it completes. The experience of designing such a hardware system will be unforgettable for me, and I will cherish the skill I gain from this course.