ECE 385

Final Project

Final Report
2-Player Battle City Game on FPGA

Haoyu Qiu, 3190110672
Xinwen Zhu, 3190110010
Source Code: https://github.com/QHY1919810/Battle-City-ECE385-final

1. Written Description of the Overview of the Circuit

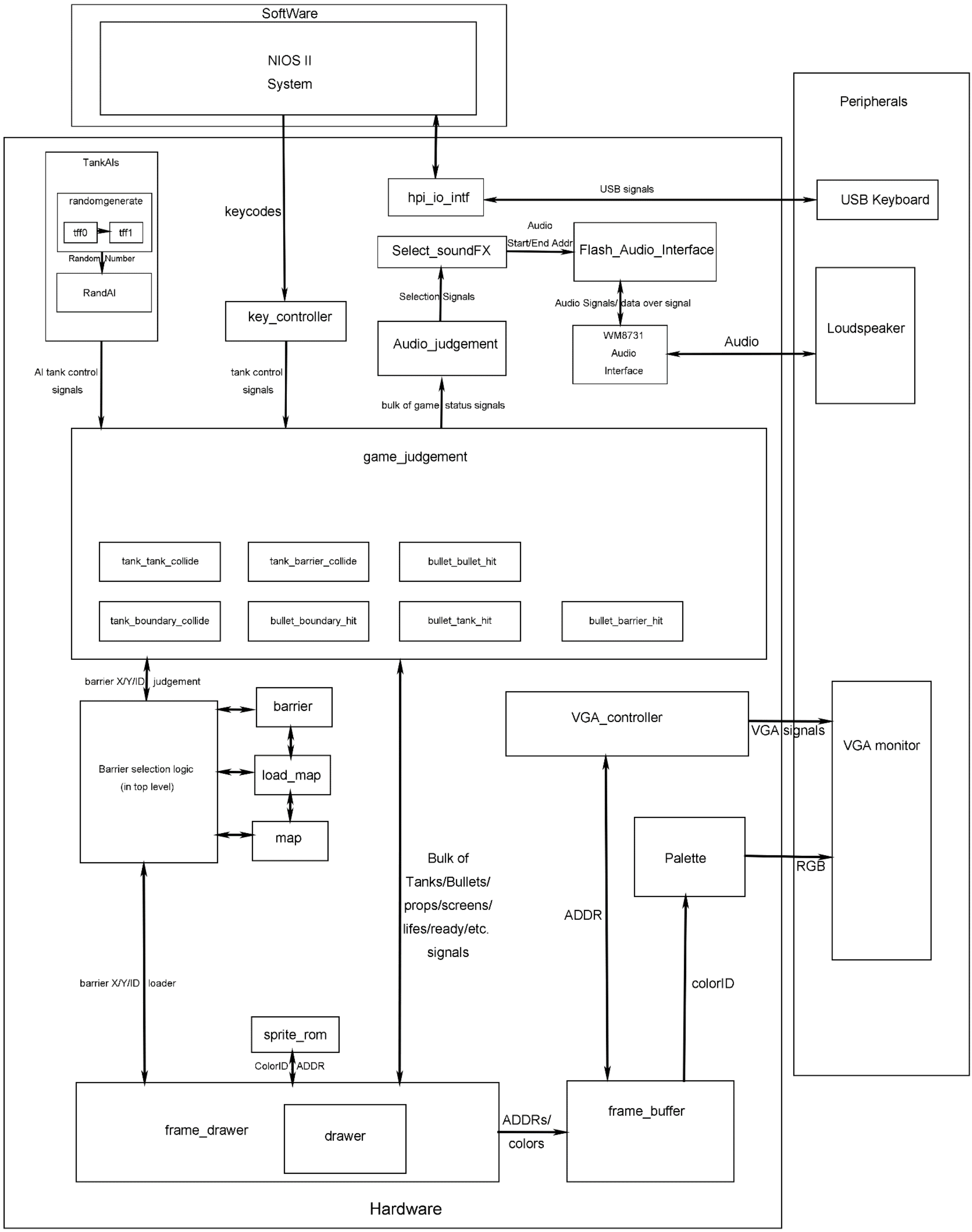
Purpose. In the final project, we implemented a classic Battle City game on the FPGA with VGA monitor, loudspeaker, and keyboard. In the game the 2 players need to destroy the enemy tanks as well as defending the flag. The map of the game containing different types of barriers, for example, destructible brick walls, river zones, forest, indestructible metal walls. When the game start, enemy tanks will be spawned on the top of the screen and will be driven by AI logic to try to destroy the base of the player, and the players will be spawned on the bottom of the screen and would need to use the keyboard to control the player tanks to shoot bullets to destroy the enemy tanks.

Feature. We modified the USB software in lab8 and make it able to support multiple keycode, player 1 use WASD to control the tank move and J to fire, player 2 use ↑↓←→ to control and numpad 5 to fire. If the player or enemy get hit by a bullet, it will be destroyed (It will be also get destroyed if one player is hit by the other player). Furthermore, the bullets can cancel out when hit another bullet. We developed 10 stages for the game, at each stage there would be 3 lives for each player, also there will be 16 enemy tanks in total and there will be at most 4 enemy tanks on the screen currently, if the players destroyed all the enemies the current stage will win. The lives of player and enemy tanks as well as the stage number will be displayed digitally on the right of the screen. Furthermore, we implemented bonus props in the game, there are 3 different types, grenade, clock, and star, the props will drop randomly when an enemy tank is destroyed. If the player gets a grenade, all the enemy tanks on the screen will be destroyed. If the player gets a clock, all the enemy tanks will get frozen for a few seconds. If the player gets a star, it will get upgraded. There are 3 levels in total, level 2 will make the tank shoot faster bullet, level 3 will make the tank be able to destroy metal wall. The levels will be remained on all stages until the player is destroyed. If so, the level will be reset to level 1. The look of tank will also change if get an upgrade. As for enemy tanks, there are also different looks, common enemy tank is just like level 1 player tank. Type-2 tank can move very fast and Type-3 tank shoot very fast bullet. This game has a high degree of completion, we added main screen, loading screen, win/loss page and their corresponding music, press "Enter" to continue and press "Esc" to back to main screen. We also added sound when any tank is destroyed. All the sounds are stored in Flash memory.

2. Written Description of the General Flow of the Circuit

As the software part, we modified the USB software and made it be able to support multiple keycode output, then the NIOS II system would output the keycodes to the hardware part. Based on these keycodes, we use the key_controller module to generate controls codes for the 2 player tanks, the 4-bits control code of a tank contains one bit fire or not, one bit is move or not and two bits of direction. As for the AI tanks, we implemented a random number generator, and we modified the probability for the operations of the AI tanks, then we generated four 4-bits control codes for the 4 enemy tanks on the screen. Also, we used random numbers to control the drop of props and the type of spawn of enemy tanks. Moreover, the barriers in the map will be loaded at the game start and the module to handles the maps and barriers will interface with the game_judgement module. All the maps, barriers, random numbers and control codes will be input to the game_judgement module, which handle the logic for the whole game, the game screens, the game status, the current maps, the tanks, the bullets, the bonus props. Also, we developed modules for the collisions between tanks, bullets and barriers, these modules are instantiated in the game_judgement module to handle those collisions. When the process of all the logics for one frame in the game_judgement module is done, it will output the ready signal and corresponding bulk of game signals to the frame_drawer module. The frame drawer will draw the current frame based on the output of game_judgement module, a drawer module is instantiated inside the frame_drawer module to handle the exactly drawing process from the sprite_rom which store all the sprites of the game as well as handling transparency. The frame will be drawn into the frame_buffer. And the VGA_controller will display the frame from the frame_buffer. We also implemented a palette to handle the colors. It will convert the ColorId_Out in the frame buffer to actual RGB colors for the VGA display in order to compress the size of images. Moreover, to handle the audio file, we store all the 8-bits audios in 8000HZ into the flash memory and we write the Flash_Audio_Interface module to read them with address of Flash and to interface with the audio_interface.vhd module as well as to decide whether to play of loop the music. Also, based on the output of bulk of game status signals from game_judgement module, we write the audio_judgement and Select_soundFX module to judge the play of audios and output the start and end address of the audio to the Flash_Audio_Interface module.

3. System Block Diagram (On Next Page)


SoftWare
NIOS II
System
Peripherals
TankAIs
randomgenerate
tff0
tff1
Random Number
RandAI
keycodes
hpi_io_intf
USB signals
USB Keyboard
Audio
Start/End Addr
Select_soundFX
Flash_Audio_Interface
Selection Signals
Audio Signals/ data over signal
key_controller
Audio_judgement
WM8731
Audio
Interface
Audio
Loudspeaker
AI tank control
signals
tank control
signals
bulk of game status signals
game_judgement
tank_tank_collide
tank_barrier_collide
bullet_bullet_hit
tank_boundary_collide
bullet_boundary_hit
bullet_tank_hit
bullet_barrier_hit
barrier X/Y/ID judgement
barrier
VGA_controller
VGA signals
VGA monitor
Barrier selection logic
(in top level)
load_map
map
Palette
RGB
Bulk of
Tanks/Bullets/
props/screens/
lifes/ready/etc.
signals
ADDR
colorID
barrier X/Y/ID loader
sprite_rom
ColorID ADDR
frame_drawer
drawer
ADDRs/
colors
frame_buffer
Hardware

4. Written Description of each Module and PIO

1. **Module: Lab8**
   Inouts: [15:0] OTG_DATA, [31:0] DRAM_DQ, AUD_ADCLRCK, AUD_BCLK, AUD_DACLRCK, I2C_SDAT.
   Inputs: CLOCK_50, KEY, OTG_INT, [7:0] FL_DQ, AUD_ADCDAT, SW.
   Outputs: [7:0] VGA_R, [7:0] VGA_G, [7:0] VGA_B, VGA_CLK, VGA_SYNC_N, VGA_BLANK_N, VGA_VS, VGA_HS, [1:0] OTG_ADDR, OTG_CS_N, OTG_RD_N, OTG_RST_N, [12:0] DRAM_ADDR, [1:0] DRAM_BA, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK, [22:0] FL_ADDR, FL_OE_N, FL_RST_N, FL_WE_N, FL_CE_N, AUD_DACDAT, AUD_XCK, I2C_SCLK.
   Description & Purpose: This module serves as the top level entity to instantiate the whole project. It instantiated all the modules we used in the project as well as the NIOS II system. And it connects all the signals of SDRAM, USB, Audio Player, VGA monitor and Flash Memory.
2. **Module: VGA_controller**
   Inputs: Clk, Reset, VGA_CLK
   Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX[9:0], DrawY[9:0]
   Description: The monitor is divided into 640 × 480 pixels, and this module will assign color for each pixel depending on whether the pixel is part of the ball.
   Purpose: This module serves as the VGA controller by getting the sync pulses and setting the parameters for each pixel.
3. **Module: hpi_io_intf**
   Inputs: Clk, Reset, from_sw_data_out [15:0], from_sw_address [1:0], from_sw_r, from_sw_w, from_sw_cs, from_sw_reset
   Outputs: from_sw_data_in [15:0], OTG_ADDR [1:0], OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N
   Description: The module passes data between the EZ-OTG chip and NIOS II. Specifically, because of the active low property, when we press Reset, OTG_RD_N, OTG_WR_N, OTG_CS_N and OTG_RST_N are all set to 1.
   Purpose: This module serves as the interface that connects the EZ-OTG chip and NIOS II.
4. **Module: sprite**
   Inputs: Clk, ADDR[19:0]
   Outputs: ColorId_Read[4:0]
   Description & Purpose: Store the data of all sprite images. (Read Only)
5. **Module: Map**
   Inputs: Clk, Map_Id[3:0], Barrier_X[8:0], Barrier_Y[8:0]
   Outputs: Barrier_Id[2:0]
   Description & Purpose: Store the data of all maps. (Read Only)

6. **Module: barrier**
   Inputs: Clk, WE, Barrier_X_In[8:0], Barrier_Y_In[8:0], Barrier_Id_In[2:0], Barrier_X_Out[8:0], Barrier_Y_Out[8:0]
   Outputs: Barrier_Id_Out[2:0]
   Description & Purpose: Store the data of barriers on the current map. (Read and Write)
7. **Module: load_map**
   Inputs: Clk, Reset, Loading, Barrier_Id_Read[2:0]
   Outputs: Load_Ready, Barrier_Write, Barrier_X[8:0], Barrier_Y[8:0], Barrier_Id_Write[2:0]
   Description & Purpose: Load the map data in "map" into "barrier".
8. **Module: frame_buffer**
   Inputs: Clk, WE, Reset, Read_AddrX[8:0], Read_AddrY[8:0], Write_AddrX[8:0], Write_AddrY[8:0], ColorId_In[4:0]
   Outputs: ColorId_Out[4:0]
   Description & Purpose: Two frame buffer swaps each 1/60 second to store/output the frame. (Read and Write)
9. **Module: palette**
   Inputs: ColorId[4:0]
   Outputs: VGA_R[7:0], VGA_G[7:0], VGA_B[7:0]
   Description & Purpose: Convert the colors with an Id into RGB format.
10. **Module: key_controller**
   Inputs: Clk, Reset, keycode_0[15:0], keycode_1[15:0], keycode_2[15:0]
   Outputs: Tank_Control_1[3:0], Tank_Control_2[3:0], Confirm, Exit
   Description & Purpose: Read the input from keyboard and convert it into control signals for tanks, Confirm ("enter") and Exit ("esc").
11. **Module: frame_drawer**

```
input logic        Clk, Reset,
input logic        Ready, // if game judgement completed
input logic        Menu, // if game is at the start menu
input logic        Loading, // if map is loaded
input logic        Over, // Game over
input logic        Win, // Game win
input logic        frame_clk, // 60Hz V-SYNC signal
// property of tanks
output logic [2:0] Tank_Id,
input logic [1:0]  Tank_Direction,
input logic [8:0]  Tank_X, Tank_Y,
input logic [2:0]  Tank_State,
// property of bullets
output logic [2:0] Bullet_Id,
input logic [1:0]  Bullet_Direction,
input logic [8:0]  Bullet_X, Bullet_Y,
```

input logic [1:0]   Bullet_State,
// Barrier buffer interface
input logic [2:0]   Barrier_Id,
output logic [8:0]   Barrier_X, Barrier_Y,
// Bonus on the map
input logic [8:0]   Bonus_X, Bonus_Y,
input logic [1:0]   Bonus_Type, // 0: not_exist, 1: upgrade, 2: stop time, 3: kill all
// Tank life
input logic [3:0]   Tank_Life_1,
input logic [3:0]   Tank_Life_2,
input logic [4:0]   Enemy_Life,
// Map id
input logic [3:0]   Map_Id,
// control signal for sprite OCM
output logic [19:0] ADDR,
input logic [4:0]   ColorId_Read,   // control signal for frame_buffer
output logic   frame,
output logic   FB_Write,
output logic [8:0]   FB_AddrX, FB_AddrY,
output logic [4:0]   ColorId_Write

Description: The color of each pixel is determined by the property of tanks, the property of bullets, props on map, barriers on map and some data for UI. When an object is needed to be drawn, this module will load parameters into "drawer". After all things are drawn, the state machine will go into an Idle state and wait for the next frame clock.

Purpose: This module decides the color of each pixel on the screen.

12. Module: game_judgment

input logic   Clk, Reset,
input logic   frame_clk,
input logic [3:0]   Tank_Control_1,
input logic [3:0]   Tank_Control_2,
input logic [3:0]   Tank_Control_3,
input logic [3:0]   Tank_Control_4,
input logic [3:0]   Tank_Control_5,
input logic [3:0]   Tank_Control_6,
input logic   Confirm, // "enter" pressed
input logic   Exit, // "esc" pressed
output logic   Tank_Die, // a tank die
output logic   Upgrade, //   a tank upgrade
// Output property of a tank according to ID
input logic [2:0]   Tank_Id,
output logic [1:0]   Tank_Direction, // 0: up, 1: left, 2: down, 3: right

output logic [8:0]   Tank_X, Tank_Y,
output logic [2:0]   Tank_State, // {2'(0: dead, 1: level_1, 2: level_2, 3: level_3), 1'movement state}
input logic   [1:0]   Random_Tank_Type,
// Output property of a bullet according to ID
input logic [2:0]   Bullet_Id,
output logic [1:0]   Bullet_Direction,
output logic [8:0]   Bullet_X, Bullet_Y,
output logic [1:0]   Bullet_State, // 0: not_exist, 1: level_1, 2: level_2, 3: level_3
// Barrier buffer interface
output logic   Barrier_Write,
output logic [8:0]   Barrier_X, Barrier_Y,
input logic [2:0]   Barrier_Id_Read,
output logic [2:0]   Barrier_Id_Write,
// Bonus
output logic [8:0]   Bonus_X, Bonus_Y,
output logic [1:0]   Bonus_Type, // 0: not_exist, 1: upgrade, 2: stop time, 3: kill all
input logic [1:0]   Random_Bonus,
input logic [1:0]   Random_Type,
// Tank life
output logic [3:0]   Tank_Life_1,
output logic [3:0]   Tank_Life_2,
output logic [4:0]   Enemy_Life,
// Map id
output logic [3:0]   Map_Id,
output logic   Ready, // ready = 1, when all judgment completed
output logic   Menu, // Menu = 1, when the game is at start menu
output logic   Loading, // Loading = 1, when the game is loading (need to press enter to continue)
input logic   Load_Ready,
output logic   Over, // Game over
output logic   Win // Game win

Description: The interaction of all objects in this game is handled in this module. All collisions/hit judgment among tanks, bullets and barriers are first handled and then output to the "frame drawer". Also, the effect of all props and success/lose conditions are handled in this module.

Purpose: This module handles the interaction between all objects and output the property of all objects to the "frame drawer".

13. **Module: tank_boundary_collide**

Inputs: Tank_X[8:0], Tank_Y[8:0]

Outputs: is_collide

Description & Purpose: Determine whether the tank collides with the map boundary.

14. **Module: tank_tank_collide**
    Inputs: Tank_X_1[8:0], Tank_Y_1[8:0], Tank_X_1_in[8:0], Tank_Y_1_in[8:0], Tank_X_2[8:0], Tank_Y_2[8:0], Tank_State_2[2:0]
    Outputs: is_collide
    Description & Purpose: Determine whether the tank collides with another state. (Allow occasional overlap)
15. **Module: tank_barrier_collide**
    Inputs: Tank_X[8:0], Tank_Y[8:0], Barrier_X[8:0], Barrier_Y[8:0], Barrier_Id[2:0]
    Outputs: is_collide
    Description & Purpose: Determine whether the tank collides with the barrier.
16. **Module: bullet_boundary_hit**
    Inputs: Bullet_X[8:0], Bullet_Y[8:0]
    Outputs: is_hit
    Description & Purpose: Determine whether the bullet hits the map boundary.
17. **Module: bullet_bullet_hit**
    Inputs: Bullet_X_1[8:0], Bullet_Y_1[8:0], Bullet_X_2[8:0], Bullet_Y_2[8:0], Bullet_State_2[1:0]
    Outputs: is_hit
    Description & Purpose: Determine whether the bullet hits another bullet.
18. **Module: bullet_barrier_hit**
    Inputs: Bullet_X[8:0], Bullet_Y[8:0], Bullet_State[1:0], Barrier_X[8:0], Barrier_Y[8:0], Barrier_Id[2:0]
    Outputs: is_hit, is_destroy
    Description & Purpose: Determine whether the bullet hits the barrier and whether the bullet destroys the barrier.
19. **Module: bullet_tank_hit**
    Inputs: Tank_X[8:0], Tank_Y[8:0], Tank_State[2:0], Bullet_X[8:0], Bullet_Y[8:0]
    Outputs: is_hit
    Description & Purpose: Determine whether the bullet hits the tank.
20. **Module: HexDriver**
    Inputs: In0[3:0]
    Outputs: Out0[6:0]
    Description & Purpose: This module serves as a HexDriver which converts binary data into hexadecimal data.
21. **Module: lab7_soc**
    Inputs: clk_clk, otg_hpi_data_in_port [15:0], reset_reset_n, sdram_wire_dq [31:0]
    Outputs: keycode_0_export [15:0], keycode_1_export [15:0], keycode_2_export [15:0], otg_hpi_address_export [1:0], otg_hpi_cs_export, otg_hpi_data_out_port [15:0], otg_hpi_r_export, otg_hpi_reset_export, otg_hpi_w_export,

sdram_clk_clk, sdram_wire_addr [12:0], sdram_wire_ba [1:0], sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n, sdram_wire_dq [31:0], sdram_wire_dqm [3:0], sdram_wire_ras_n, sdram_wire_we_n
Description & Purpose: Soc of Lab8

22. Module: vga_clk
Inputs: inclk0
Outputs: c0
Description & Purpose: Transform a clock signal of 50MHz into 25MHz for VGA controller.

23. Module: ocm_clk
Inputs: inclk0
Outputs: c0
Description & Purpose: Transform a clock signal of 50MHz into 200MHz.

24. Module: audio_interface
Inputs: clk, reset, INIT, LDATA, RDATA, AUD_BCLK, AUD_ADCDAT, AUD_DACLRCK, AUD_ADCLRCK, I2C_SDAT, I2C_SCLK
Outputs: INIT_FINISH, adc_full, data_over, AUD_MCLK, AUD_DACDAT, ADCDATA
Description & Purpose: This is an interface for the audio hardware based on I2C protocol.

25. Module: Flash_Audio_Interface
Inputs: Clk, Reset, data_over, Audio_Reset, play, loop, [22:0] Start_Addr, [22:0] End_Addr, [7:0] FL_DQ
Outputs: FL_OE_N, FL_RST_N, FL_WE_N, FL_CE_N, [22:0] FL_ADDR, [15:0] Audio_Data, End_flag
Description: Use a state machine to handle the play of audio. This module read 8000HZ 8-bits audio file from flash memory and feed it into the audio interface depend on the data over signal return from the WM8731 audio interface.
Purpose: This module interfaces with the flash memory and the WM8731 audio interface.

26. Module: SoundFX_Selector
Inputs: Clk, [17:0] InputSelect,
Outputs: loop, play, Audio_Reset, [22:0] Start_Addr, [22:0] End_Addr
Description & Purpose: This module will return the corresponding audio file and its settings with the one-hot code InputSelect signal. When the InputSelect change, Audio_Reset signal will be 1 and tell the Flash_Audio_Interface module to stop current audio and switch to new one.

27. Module: Audio_judgement
Inputs: Clk, End_flag, Tank_Die, Win, Upgrade, Menu, Loading, Over, [3:0] Tank_Control_1, [3:0] Tank_Control_2, [1:0] Bonus_Type,
Outputs: [17:0] InputSelect

Description & Purpose: This module takes the bulk of input from game_judgement and use a state machine to control whether and which audio to play.

28. **Module: tff0**
    Inputs: t, c
    Outputs: q
    Description & Purpose: This module generates the first bit for the random number.
29. **Module: tff1**
    Inputs: t, c
    Outputs: q
    Description & Purpose: This module generates the following bits after the first bit of the random number.
30. **Module: randomgenerate**
    Inputs: Clk
    Outputs: [30:0] o
    Description & Purpose: This module generate a 31-bits random number by instantiating tff0 and tff1 modules.
31. **Module: RandAI**
    Inputs: Clk, Reset_h, [31:0] countervalue, [4:0] randominput1, [1:0] randominput2.
    Outputs: [3:0] AI_tank_control.
    Description & Purpose: This module takes random numbers and generate control codes for the AI tanks with adjusted probability for different behaviors.
32. **Module: TankAIs**
    Inputs: Clk, Reset
    Outputs: [3:0] AI_tank_1, [3:0] AI_tank_2, [3:0] AI_tank_3, [3:0] AI_tank_4, [23:0] Random_others
    Description & Purpose: This module instantiates the RandAI and randomgenerate modules to generate control codes for all 4 AI tanks, and the remaining random number will be also output for the other modules to generation of props and enemy tank types.
33. **PIO: keycode_0**
    Inputs: clk, reset
    Outputs: keycode_0_export [15:0]
    Base Address: 0x0000_0030
    Description: The data of the first two pressed keys
    Purpose: Take the data from keyboard and pass it to FPGA.
34. **PIO: keycode_1**
    Inputs: clk, reset
    Outputs: keycode_1_export [15:0]
    Base Address: 0x0000_0020
    Description: The data of the second two pressed keys
    Purpose: Take the data from keyboard and pass it to FPGA.

35. PIO: keycode_2

Inputs: clk, reset

Outputs: keycode_2_export [15:0]

Base Address: 0x0000_00a0

Description: The data of the third two pressed keys

Purpose: Take the data from keyboard and pass it to FPGA.

36. PIO: otg_hpi_address

Output: otg_hpi_address _export [1:0]

Base Address: 0x0000_0090

Description: The signal set the hpi registers that we are doing operation on

Purpose: Depending on the operation, the signal will be set to different value.

37. PIO: otg_hpi_data

Input: otg_hpi_data_in_port [15:0]

Output: otg_hpi_data_out_port [15:0]

Base Address: 0x0000_0080

Description: The 16-bits data that will be placed into hpi-data register

Purpose: We need the PIO to transfer data between USB chip and NIOS II.

38. PIO: otg_hpi_r

Output: otg_hpi_r_export

Base Address: 0x0000_0070

Description: otg_hpi_r will be set to 1 when read data from hpi registers.

Purpose: Enable IO_Read.

39. PIO: otg_hpi_w

Output: otg_hpi_w_export

Base Address: 0x0000_0060

Description: otg_hpi_w will be set to 1 when write data to hpi registers.

Purpose: Enable IO_Write.

40. PIO: otg_hpi_cs

Output: otg_hpi_cs_export

Base Address: 0x0000_0050

Description: otg_hpi_cs will be set to 0 when we want to IOWrite or IORead.

Purpose: Enable IOWrite and IORead.

41. PIO: otg_hpi_reset

Input: reset_reset_n

Base Address: 0x0000_0040

Description: otg_hpi_reset will be set to 1 when we reset.

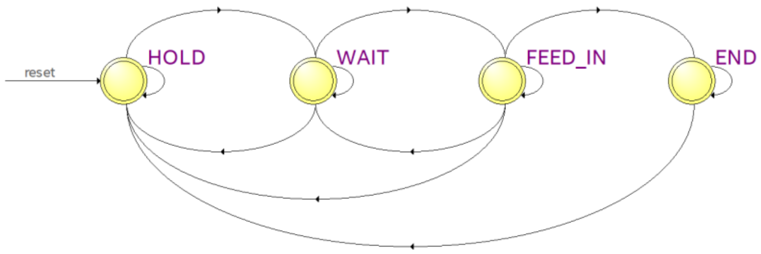
Purpose: Enable reset.

5. Design Procedure

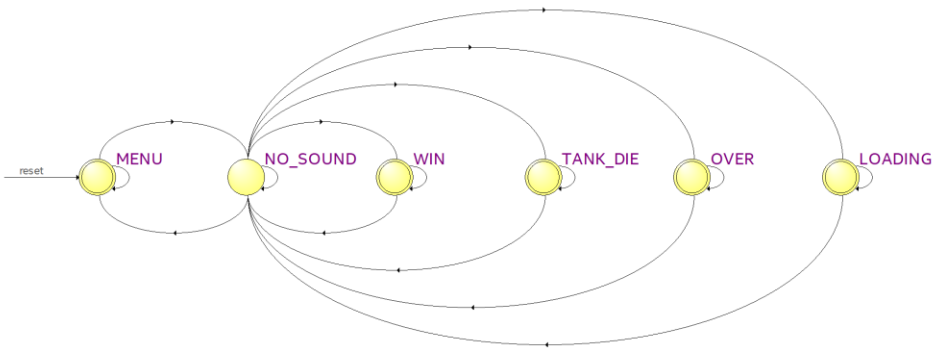
We used lab8 as the foundation of this project with its USB keyboard controller and VGA controller. At the first and the second week, in order to implement 2 players game, initially we tried to use a ps/2 keyboard and we successfully make it to run n-key roll over properly, however, we found that the ps/2 keyboard was conflict with the WM8731 audio interface and make our game significantly slower. Hence, we then decided to modify the original USB software code in NIOS system in order to make it able to support 6 keys.

After that, in order to implement audio, we looked up some tutorial of I2C protocol and we added sound into the project successfully. To make it fast enough to play the audio and for convenience, we used flash memory to store the audio which could store the audio file permanently. We used two state machines to control the audio, one is in Flash_Audio_Interface module, the other is in Audio_judgement module, the first is to fetch audio from the Flash Memory and decide whether to feed data into the audio interface depend on the data over flag, whether to stop and whether to loop the audio.The second State machine is to control the current audio to play.

State Machine in module Flash_Audio_Interface:

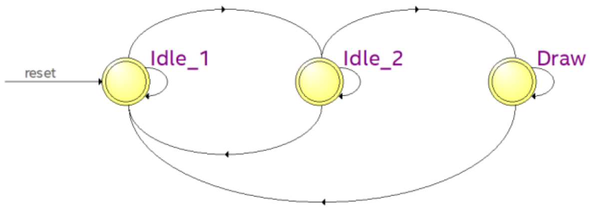


State Machine in module Audio_judgement:

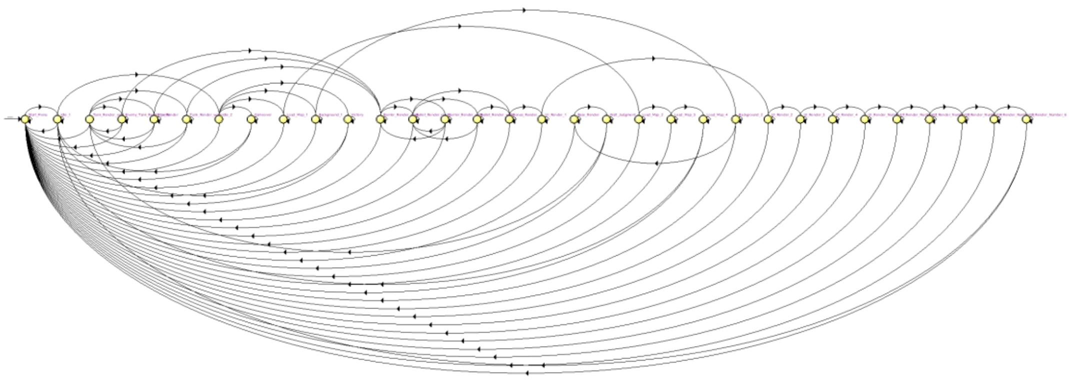


Then, we implemented the game judgement and the frame drawer logic, in the drawer module inside the frame drawer, we also implemented state machine to handle the frame drawing process.

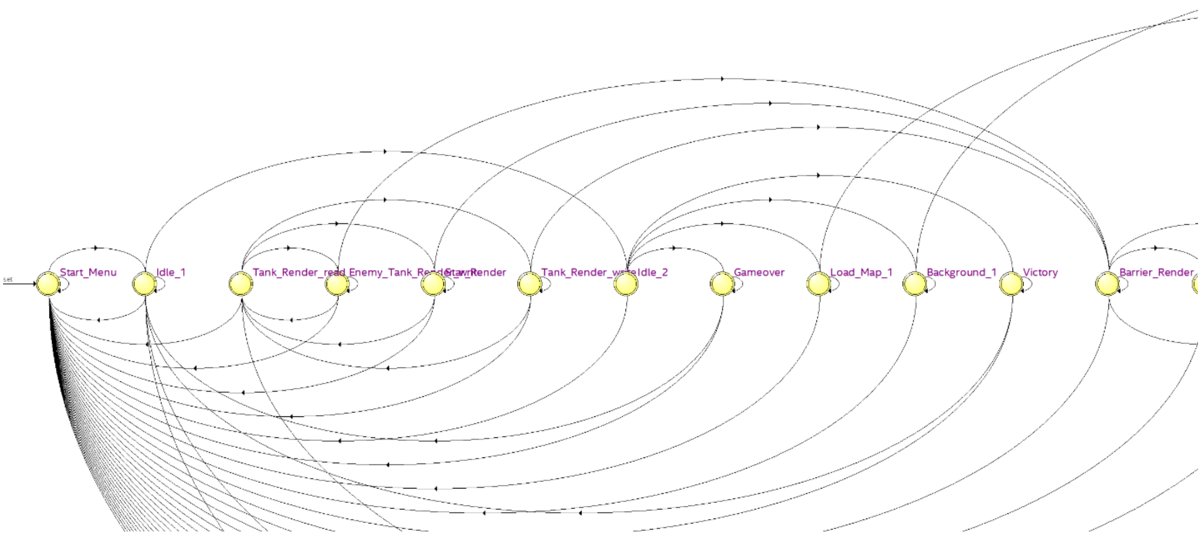
State Machine in module drawer:

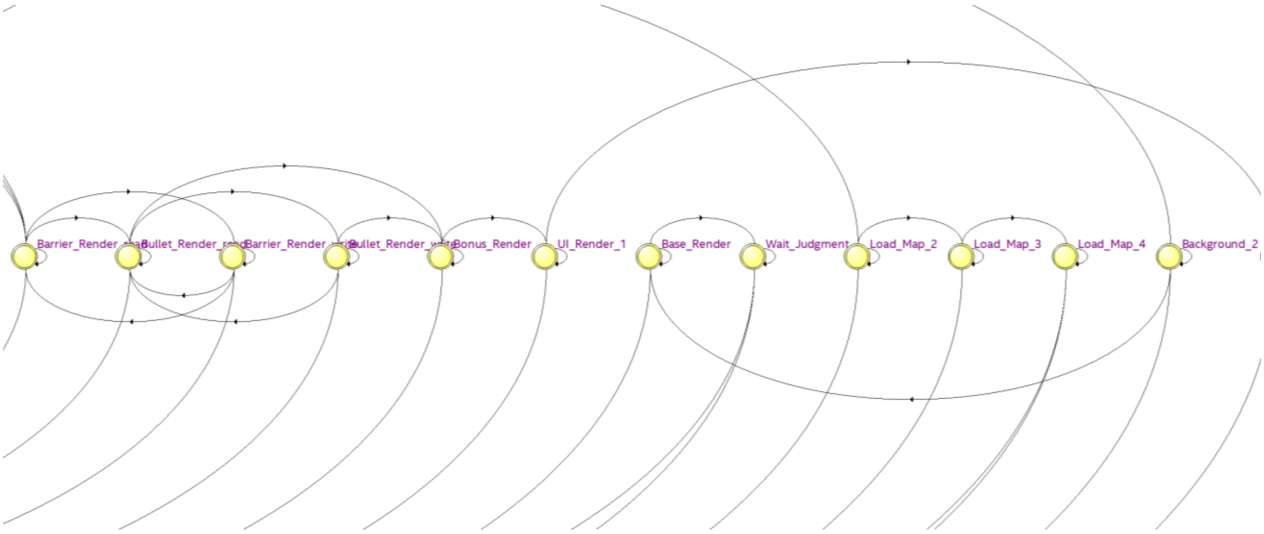


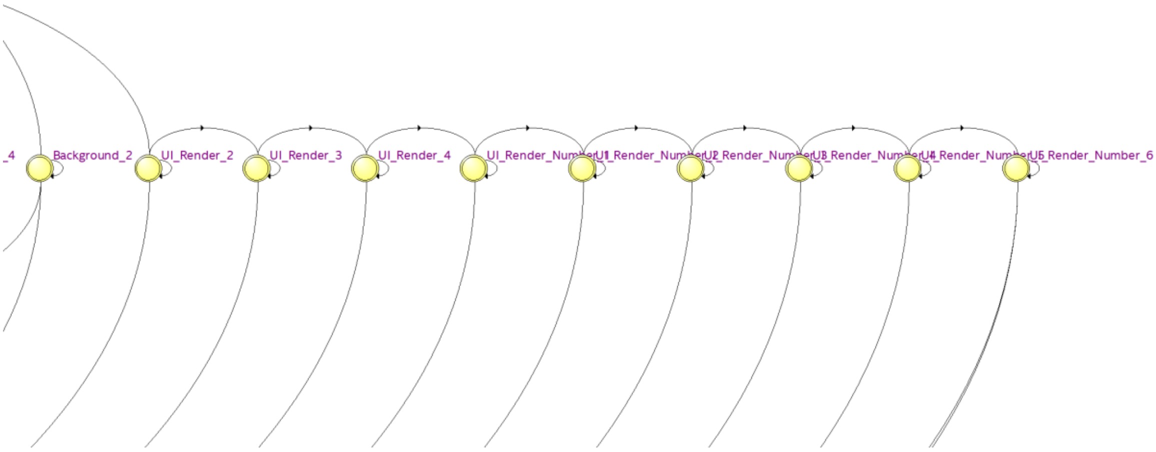
In the frame_drawer module, we implemented a state machine select all the objects like tanks, barriers, UI digital number, game screens, etc. to draw. When an object is needed to be drawn, the frame_drawer will load parameters into the drawer module. After all things are drawn, the state machine in the frame_drawer will go into an Idle state and wait for the next frame clock.
State Machine in module frame_drawer:



Detailed Diagram for State Machine in module frame_drawer:

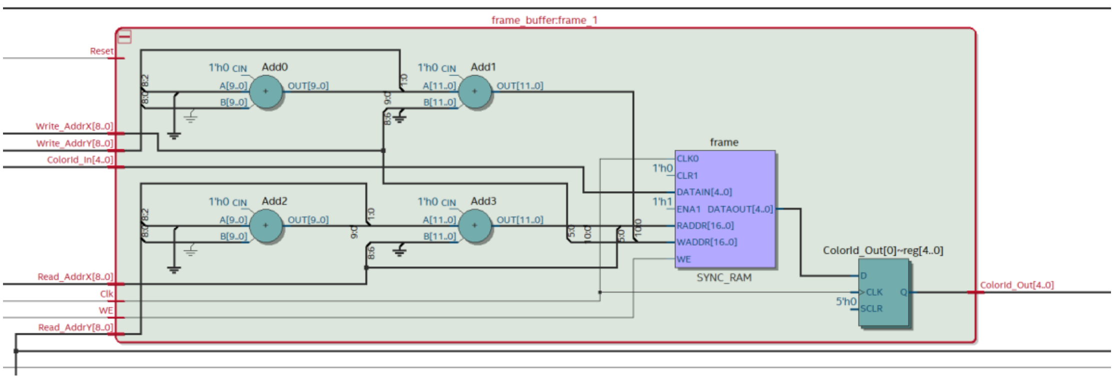




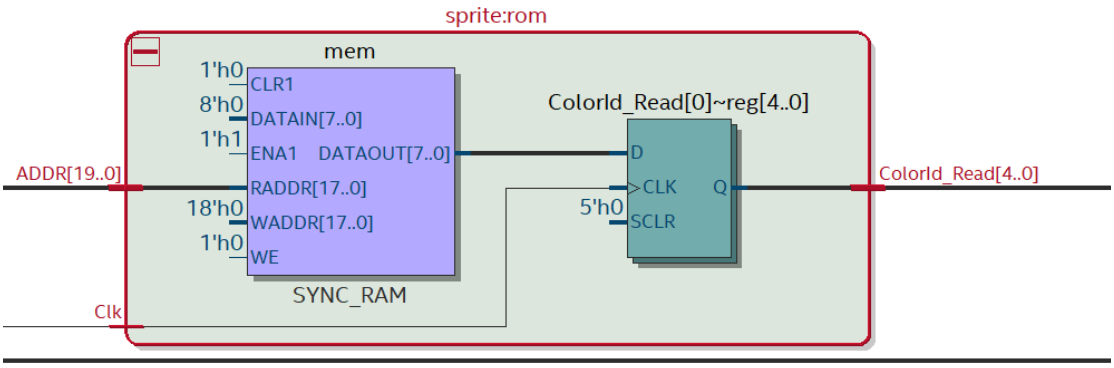


We store all the sprites and the maps and barriers of the game by creating ROM on the On Chip Memory to guarantee fast reading speed. We also implemented frame buffer on the On Chip Memory. For higher performance, we use 200MHZ clock for the OCM.

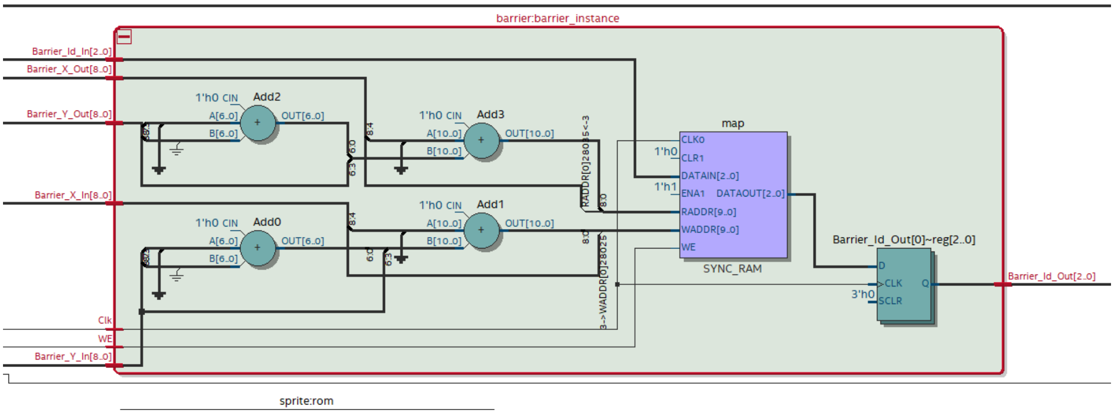
Block Diagram for frame buffer:



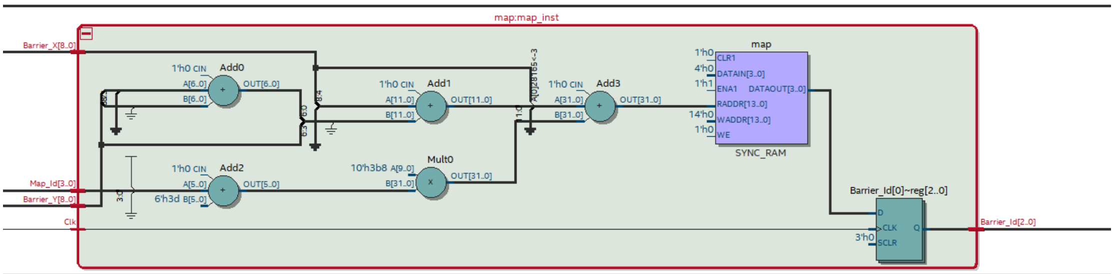
Block Diagram for sprite ROM on the OCM:



Block Diagram for the barrier module on the OCM:



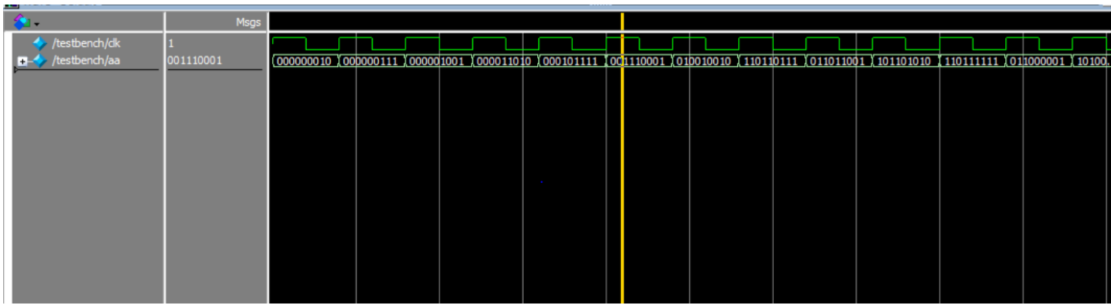
Block Diagram for the map module on the OCM:



We used python tools to convert images into txt file as well as creating color palette. Moreover, as we need to create 10 maps, for convenience, we developed a tool in python to convert 34x28 image which containing colors corresponds to the types of the barriers directly into readable game maps. As for the audio file, we stored it on the Flash Memory, we also developed a python tool to convert all the audios into 8000HZ 8-bits audios and packed all the audios into one audio file in ".ram" format in order to write into the Flash Memory directly. The Python codes are on https://github.com/QHY1919810/Battle-City-ECE385-final.

As for the control for the enemy tanks, we generate random number first and adjust probability for the behaviors of the enemy tanks. In order to generate random number, we use the tff0 module for the first bit and tff1 module for the other bits. We tested the algorithm and to make sure that it is able to generate a random number.

Here is a simulation wave form for a 9-bits random number generator.



6. Design statistics and Discussions

Design Resources and Statistics Table	
LUT	6817
DSP	0
Memory (BRAM)	2414592
Flip-Flop	3141
Frequency	129.43MHz
Static Power	105.67mW
Dynamic Power	0.70mW
Total Power	212.96mW

During the debugging phase, we found the following several bugs which are interesting:

1. We found that the object on the screen is torn, this is because the On Chip Memory is not fast enough, when we accelerate the clock for OCM to 200MHZ, this bug get fixed.

2. We found that the audio is playing far too fast, this is because the data over signal returned form the WM8731 audio interface is not a purely one clock impulse, when we add a data over flag to detect the rising edge of this signal, the bug get fixed.
3. We found that the game is very slow and the control signal of tanks will flip randomly when combining ps/2 keyboard and audio player, this is because the ps/2 keyboard conflict with the WM8731 audio interface, when we switch to USB keyboard, this bug get fixed.

7. Conclusion

In the final projects, we successfully implement a 2-players classic Battle City game on the FPGA, we managed to implement all the feature that we stated in the proposal. There are 10 stages in the game and the feature in the game is rich. The game we implemented in our project is well playable and is of high completion level. We managed to apply many techniques when implementing this game, for example, use of frame buffer, color palette, use of flash memory, processing of audio. And we gain rich experience and get deeper understanding of FPGA and different types of the hardware through this final project.