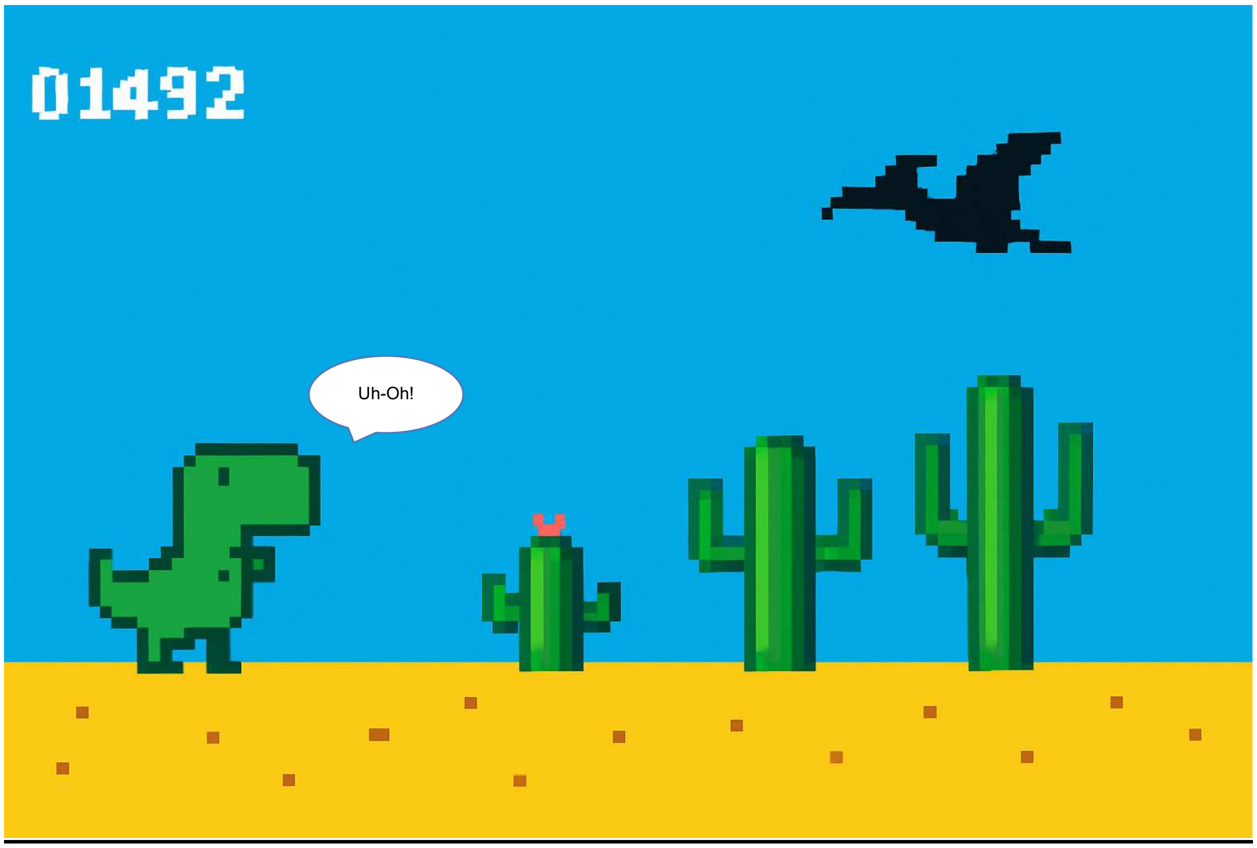
DINO RUN

Design Document for CSEE 4840 Embedded System Design

Swapnil Banerjee (sb5041), Roshan Prakash (rp3187), Anne Rose Sankar Raj (as7525)



Contents

Introduction:

In this project we will aim to recreate the Dino Run game on the FPGA, adding several twists to enhance gameplay. The graphics and background have been overhauled to enhance the visual experience. The game will consist of basic character movement, powerup activations, obstacle detection, and a running score system. We will use an up-arrow key to make the dino jump and a down arrow to make the dino duck. The game will end when the character collides with an obstacle. Dino will gain different abilities from the powerups that it collides with. The Godzilla powerup will make the dino invincible for a certain time. The Slow Mo powerup will slow down the game speed. The Shrink power-up will make the dino tiny such that it can avoid the pterodactyls. The game will run entirely on FPGA hardware and a VGA display.

Outline:

Input Module:

- Interfaces with the De1-Soc
- Reads:
    - Directional Input (Up, Down)
- Translates the inputs into higher level commands
    - Jump
    - Duck

Game Module:

- 2D scrolling: The dino will stay on the left of the screen during movement
- Gravity physics
- Power up logic
- Obstacle logic:
    - Spawns and updates the obstacles
    - Stores the active states
- Collision Detection:
    - The Dino will be blocked by the ground.
    - The Dino will interact with enemies and power ups that will execute their interaction after contact with the dino.
- Game States:
    - Playing, Game over, reset
- Increments and displays the score

Display Module:

- Sprite Buffer
- VGA signal generation
- Frame rendering

VGA Output:

- Uses ADV7123 DAC in order to generate analog VGA signal
- Sends visuals to monitor

Animation: Score animation, jumping animation, death animation.

Audio Output: The game will have music on start and audio effects (noise on crash).

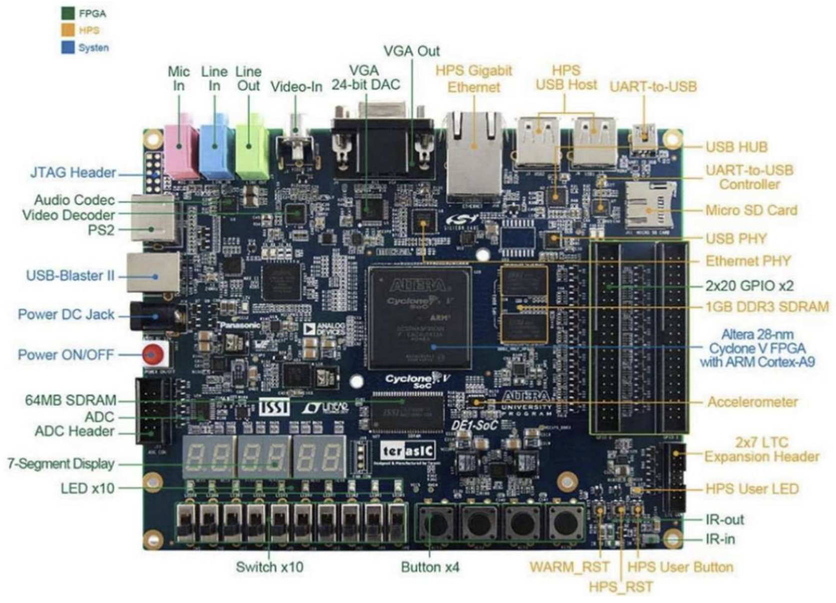
I/O Device:

Video Output: VGA

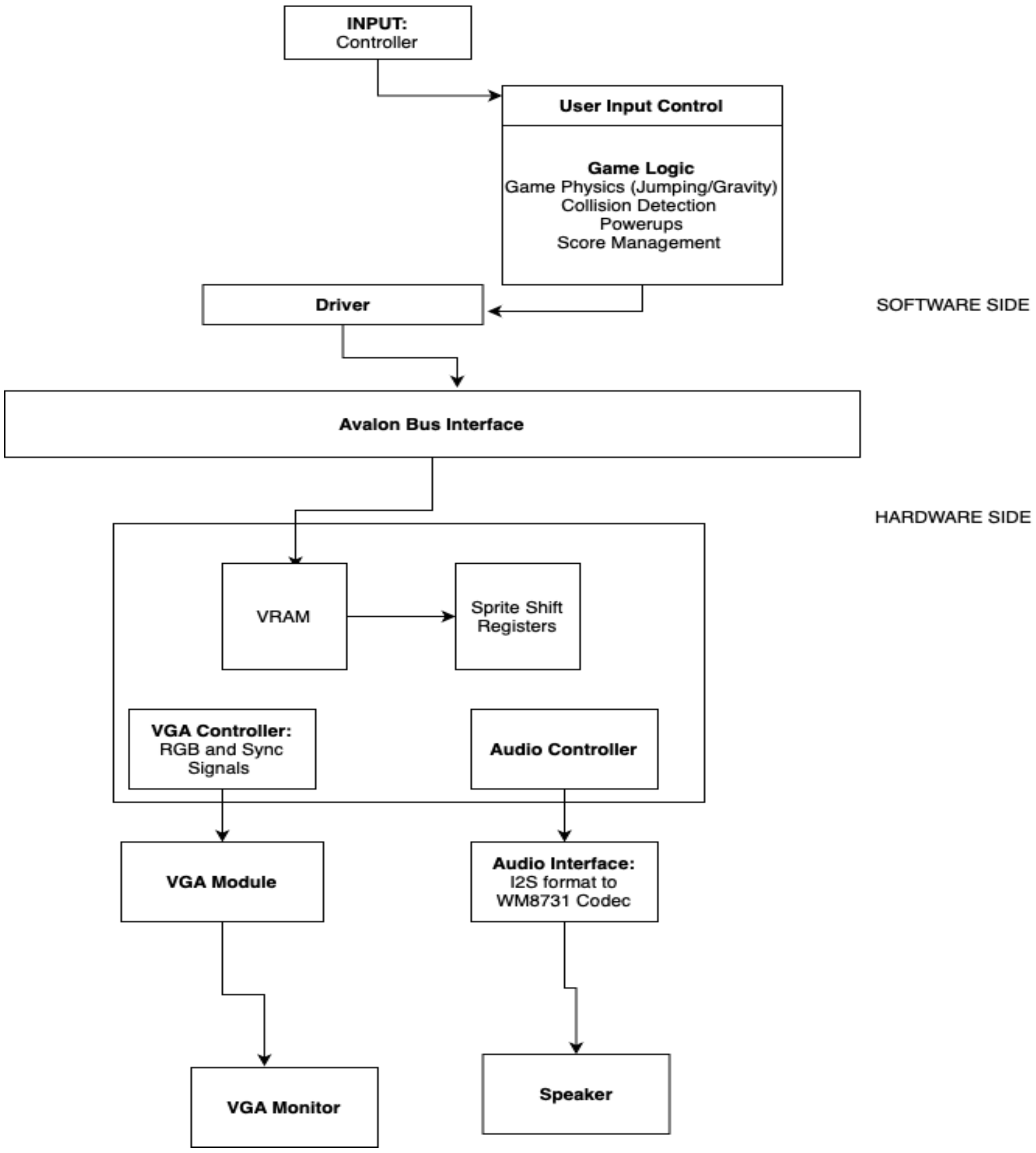
Audio Output: Speaker

Controller Input: Joystick

Platform:



Basic Block Diagram:



We use the HPS to handle sequential logic. For example, we handle the jumping and the score in c-code. We use the FPGA to render the sprites and to check pixel collisions in parallel.

The Avalon Bus is essentially the connection between our Verilog and c-code.

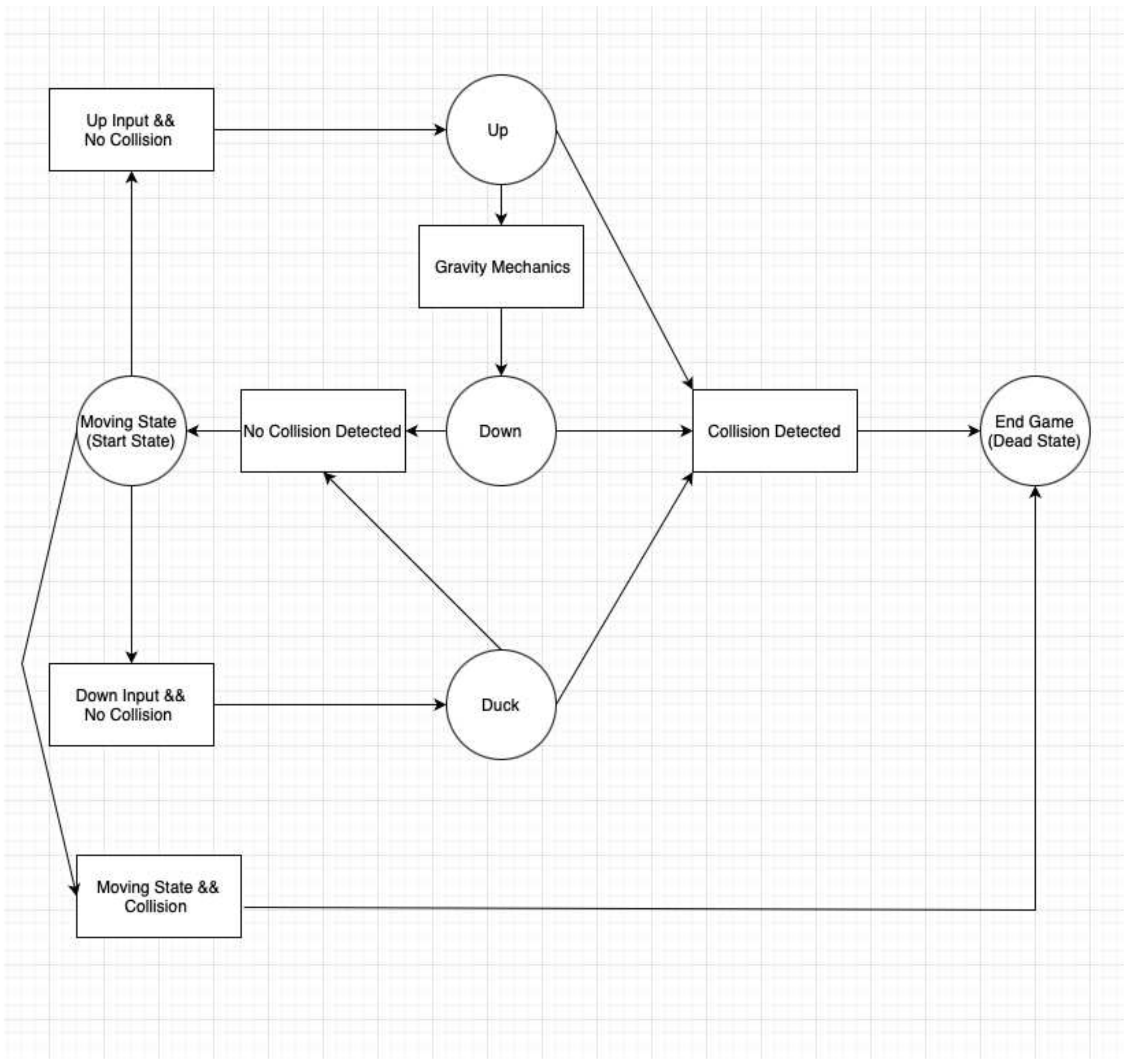
Movement Logic:

Top Edge: Collision Detection

Bottom Edge: Collision Detection

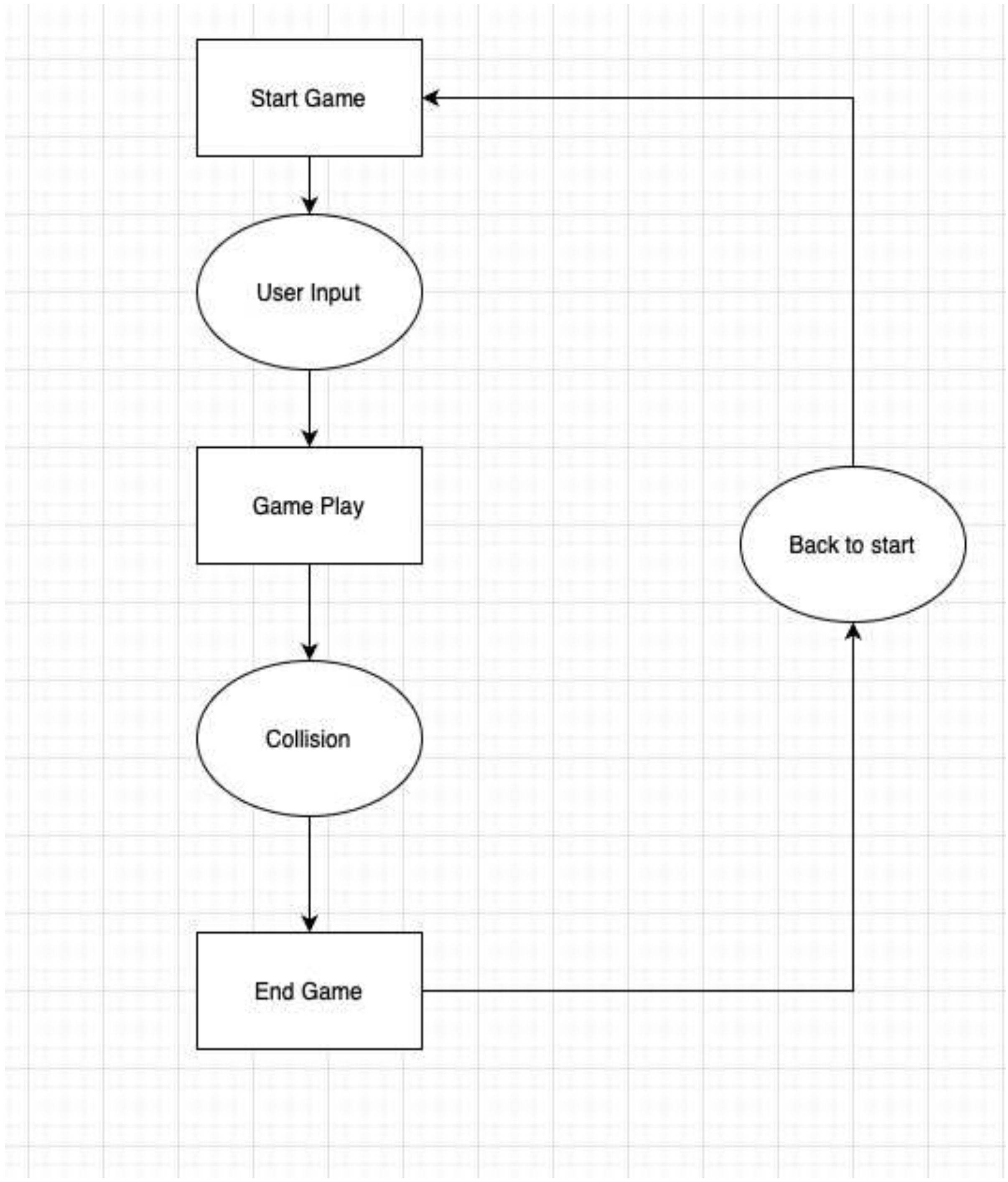
Right Edge: Collision Detection

Left Edge: No Collision Detection



Gameplay Logic:

```
Game{


while(1):

draw_obstacles();
move_obstacles();


   IF User Presses Jump Key:
        jump()
   IF User Presses duck Key:
        duck()
   IF collision detection==true:
        IF powerup:
            switch(case):
            Case ‘Godzilla’:
                        while(300):
                                    Change sprite to godzilla
                                    Remove obstacle sprites on its way

            Case ‘Slow-mo’:
                        while(300):
                                    Game-speed=Gamespeed-1
                         powerup=1
            Case ‘Shrink’:
                        while(300):
                                    Reduce sprite size
                                     Power_time = 300


           IF Obstacle collides with Dinosaur:
           END Game()

   update_score()

}
Display "Game Over" Screen
Display Final Score
Offer Restart or Quit Option

}
```


Start Game
User Input
Game Play
Collision
End Game
Back to start

Display Logic:

Our plan is to render the images by converting the PNG images to HEX files and reading them from rom. To prevent loading up the memory, we have estimated how much memory each image would potentially take up and we also came up with logic for the sprite creation to save up memory.
We will design a VGA controller to generate the required CGA signals from the FPGA. It will read pixel data from different sources depending on the current state of the game. It will also include horizontal sync, vertical sync signals. This controller an Implementation will be done using all necessary signals to control and render the VGA monitor. Screen resolution is 680*480 with 60Hz.

Audio Logic:

The de1soc comes with a Wolfson codec audio chip for handling 24-bit audio. This chip contains 2 ADCs and 2 DACs to interface with analog jacks and the FPGA with a digital interface. We will configure it via the I2C bus and provide a clock rate of 12MHz. We will only use of the DAC channels of the codec to transmit the signal for our Audio speaker. Altera provides two IP blocks for use to control the audio.

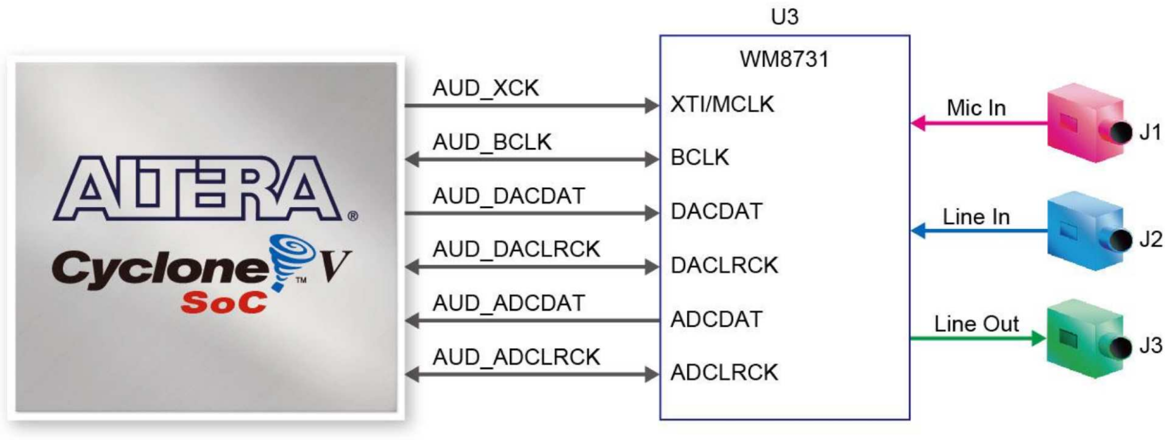
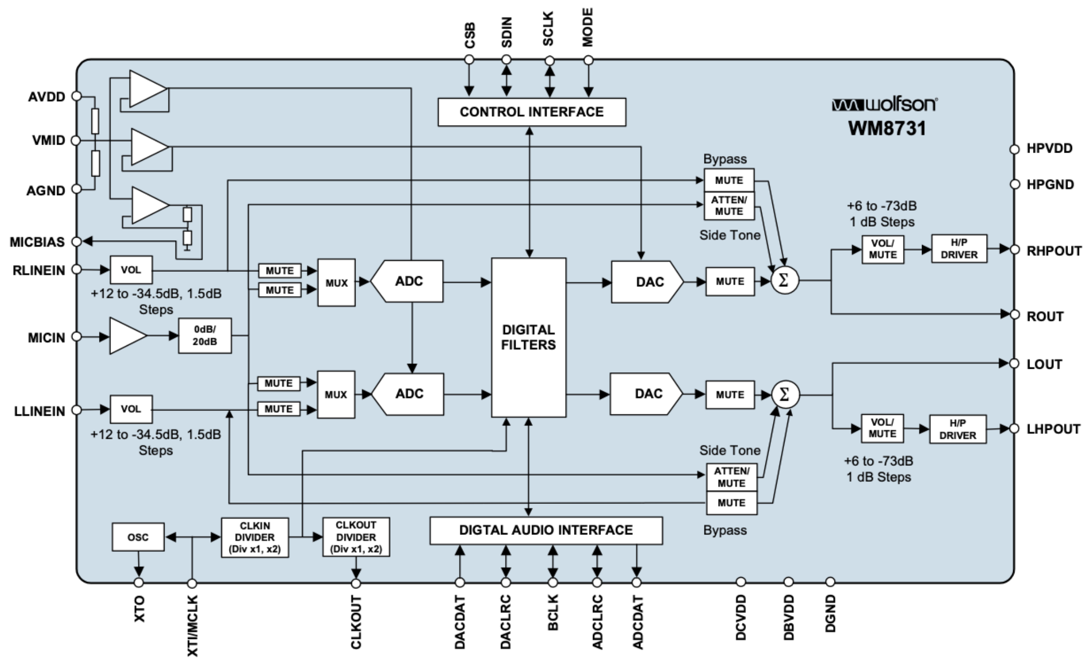


Fig: Wolfson chip connections with the board

BLOCK DIAGRAM



4.1 Register Map

Device drivers control and communicate with the Audio Core through four 32-bit registers. By writing or reading these registers, data can be fetched from the ADC or sent to the DAC. Table 1 shows the format of the registers.

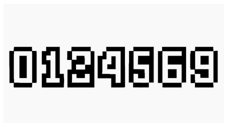
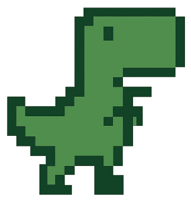
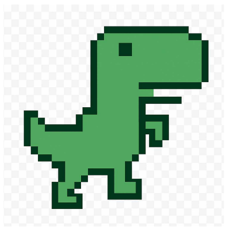
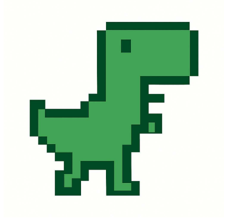
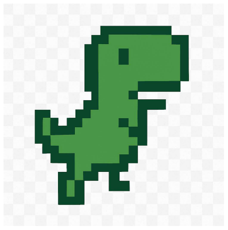
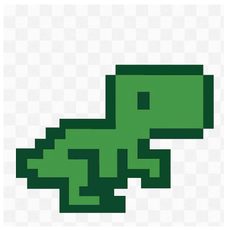
Table 1. Audio Core Register Map

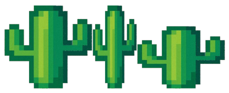
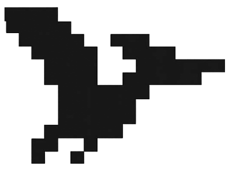
Offset in bytes	Register Name	R/W	Bit Description									
			31…24	23…16	15…10	9	8	7…4	3	2	1	0
0	`control`	RW	*(1)*			WI	RI	*(1)*	CW	CR	WE	RE
4	`fifospace`	R	WS LC	WS RC	RA LC			RA RC				
8	`leftdata`	RW *(2)*	Left Data									
12	`rightdata`	RW *(2)*	Right Data									

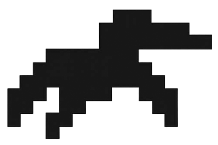
Player Input:

The logic for input will be based on using libsub to receive and decode the button presses on the us- connected joystick.

Sprites:

Category	Graphics	Size (bits)	Number of images	Total size (bits)
Score		8*80(8*8 for each digit)	1	24*8*80 = 15360
Dino	  	40*50(each dino)	3	24*3*40*50 = 48000
Dino Jump		40*50	1	24*40*50 = 48000
Dino Duck		40*50	1	24*40*50 = 48000

Dino Dead		40*50	1	24*40*50 = 48000
Godzilla		100*80	1	24*100*80 = 192000
Small Cactus		30*20	2	24*2*30*20=28800
Cacti Together		150*40	1	24*150*40=144000
Lava		40*30	1	24*40*30=28800
Powerup		20×30	3	24*3*20*30=43200
Pterodactyl		40*50	2	24*2*40*50 = 48000

				
Total				836160

Audio:

	Time	Fs (KHz) 12 (9 bits)	Total size (bits)
Dino Die	3	12	9*(3*8000) = 216000
Jump	1	12	9*(1*8000) = 72000
Background Music	1	12	9*(1*8000) = 72000
Dino crash with cactus	1	12	9*(1*8000) = 72000
Dino crash with pterodactyl	1	12	9*(1*8000) = 72000
Dino crash with lava	1	12	9*(1*8000) = 72000
Dino crash with Powerup	1	12	9*(1*8000) = 72000
Total			648000

Total bits required is 1,484,160.

We know that total bits provided by the de1-soc is 4,450 Kbits. Our memory requirement is only 1,484 Kbits. So initial design should fit well with the provided resources.

Address Mapping:

Address	Name	Description
0	dino_position_y	X position is fixed
1	score	Dino's score

2	dino_knocked	If dino collides with obstacle then dino_knocked == 1 else 0
3	background_color_R	R background color
4	background_color_G	G background color
5	background_color_B	B background color
6	console_command	If command/input received, then console_command == 1 else 0
7	obstacle_position_x	Y position will be fixed for the obstacles

Milestones:

Milestone1:
Implement the sprites with dino, powerups and all the obstacles.

Milestone2:
Implement the joystick control and the movement logic for the dino.

Milestone3:
Implement the entire game logic with audio output.