TOP GUN

CSEE 4840 EMBEDDED SYSTEMS - SPRING 2023

PROJECT MEMBERS:

APARNA MURALEEKRISHNAN (am5964)

KURALOVIYAN MAHENDRA SENTHILNATHAN (ks4065)

EASHAN SAPRE (es4069)

DESIGN DOCUMENT

1. INTRODUCTION

We attempt to bring the Top gun flight game to the FPGA board. The player operates the fighter jet in a side-scrolling game while attempting to avoid mountains and missiles. If the player's aircraft touches any of the obstacles on screen, the game ends. The fighter jet ascends each time the player presses a key, else it descends downward due to gravity. The player earns points based on the distance covered by the jet. A high score is maintained and displayed during all game play.

2. SYSTEM OVERVIEW

The major components in our design are shown as follows:

(a) game controller, (b)user input, (c) video module and, (d) audio module

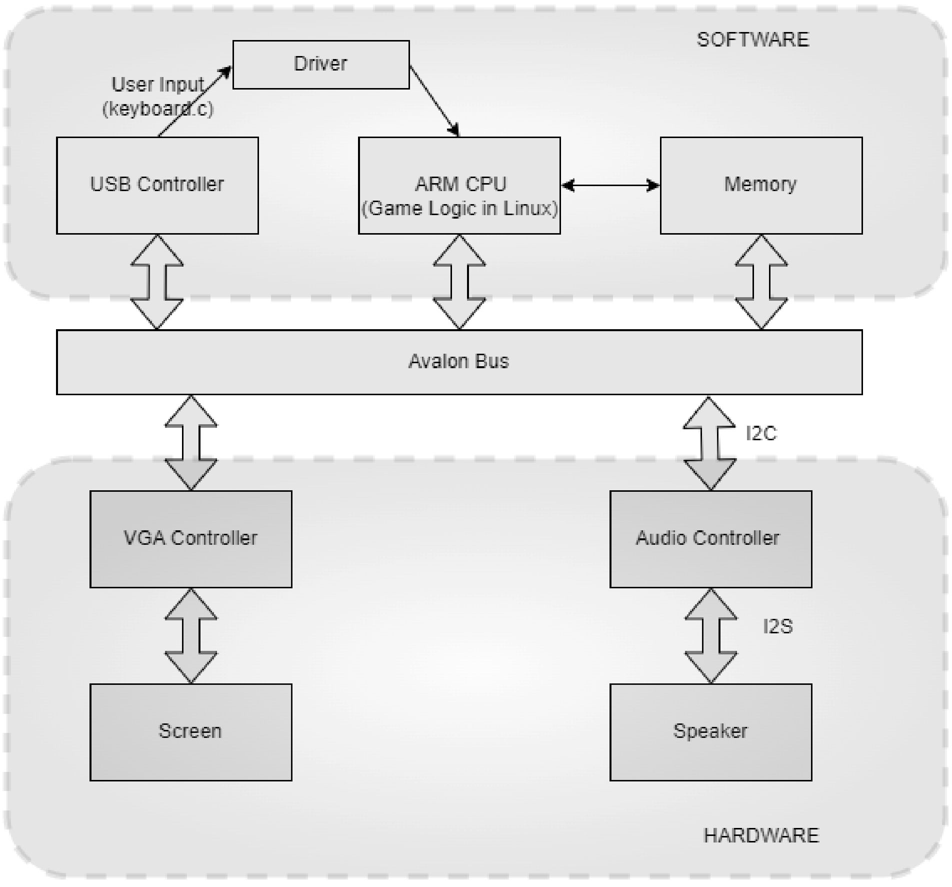


Fig 1: System Block Diagram highlighting hardware and software components

3. GAME RULES:

- The game starts when the ”Start” button is pressed, the game can be paused by another button press.
- The player controls the jet via keyboard input, pressing the spacebar to cause the jet to ascend vertically. The player has no control over the horizontal coordinates of the jet.
- Points can be scored by successfully maneuvering around obstacles.
- Crashing into obstacles causes the game to end. High score is updated if the player beats the previous record.
- There will be three difficulty levels, with more missiles/mountains appearing on the screen as points increase above set thresholds. The screen scroll speed also increases with increasing difficulty.

4. GAME CONTROLLER:

4.1. Game Logic:

This is the core submodule of the game logic which interfaces with all of the other submodules, instructing them what to do based on the game rules. The game should constantly update the screen by supplying the graphic generator with the location of the fighter jet, the number of mountains and frequency of missiles generated based on the score, as well as the judgment on whether the game is over. This module also updates the difficulty level and decides the scroll speed and obstacle frequency accordingly. We will implement a state machine as the core control logic in this module.

4.2. Tracker:

The function keeps track of the movement of the jet by the previous location, instructions from the keyboard and also a sub function that controls the automatic descent of the aircraft. Once the spacebar button is pressed, the jet should ascend with a vertical upward speed. When the spacebar is not pressed, the aircraft descends due to the gravitational pull. We only care about the vertical location of the aircraft. (The horizontal speed remains constant for a difficulty level and is updated automatically). Descend control: $Y_{jet}=(v*t)-[(1/2)*g*t^2]$

4.3. Judge:

This function calculates the real-time score. The function will compare the X/Y coordinates of the fighter jet with that of all obstacles. The speed of the obstacles remain constant and we use a counter to calculate the score. As long as the game is not over, the counter increments at the positive edge of a clock cycle depending on the distance between adjacent obstacles, and the speed of screen scroll. The score appears on the top right of the screen. This function also determines whether the game is over or not. If the coordinates of the jet overlaps with any obstacle, the counter stops. If so, the jet blasts (graphically) and the submodule will send a message to the graphic controller, to generate the "Game over" screen with the final score and a "Restart" button.

4.4. Obstacle Generator:

This function calculates the length of the mountain that will soon arrive from the right side of the screen as well as the X and Y coordinates of the existing mountain's on the screen. It also calculates the placement of missiles in the sky. The mountain/missles's placement would continue to move to the left as the screen scrolls. As long as the space between the mountains remains constant, the length of the mountain's should be random (within a specified range). The missiles also fly at random heights (the X coordinates remain unchanged, like the height of the mountain) within a specified height range.

4.5. Graphic generator :

This submodule receives all data required for the generation of graphics, including the coordinates of the jet and the mountains and missiles and the signal whether the game is over or not. These coordinates are stored in memory and updated according to the game logic. The graphic controller uses addressing to access memory-mapped data and then displays the necessary graphics components on the screen.

4.6. Audio generator:

The audio sounds needed in the game, including the background music, the Top Gun Anthem music, as well as the sound when the jet crashes, are encoded inside the audio generator. This submodule should instruct the audio controller which one to play, based on the game logic.

5. VGA Block

5.1. VGA controller:

The DE1-SoC board includes a 15-pin D-SUB connector for VGA output. The VGA synchronization signals are provided directly from the FPGA, and the Analog Devices ADV7123 triple 10-bit high-speed video DAC (only the higher 8-bits are used) is used to produce the analog data signals (red, green, and blue). The following figure, taken from the FPGA manual, gives the reference schematic:

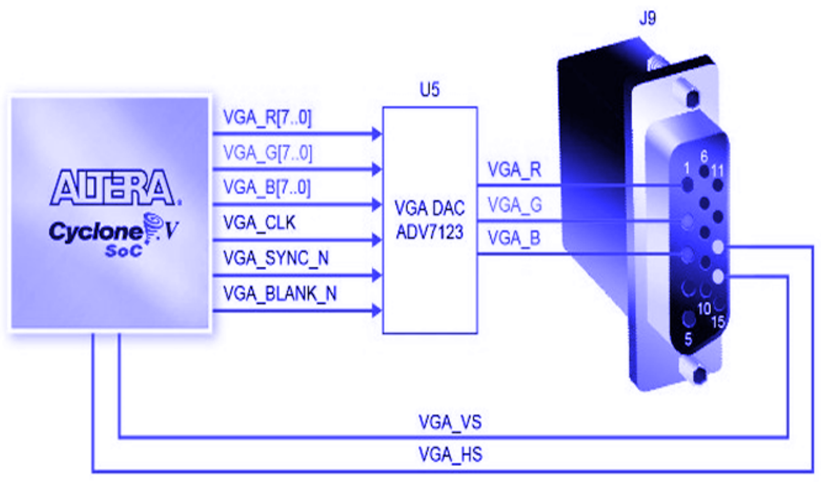


Fig 2: VGA Interface

5.2. Sprite controller:

The game logic calculates and stores input to be fed to the sprite controller: the X and Y coordinate of different elements. Then the sprite controller will send the RGB values of each pixel to the VGA controller. Our game uses seven types of elements that will be displayed on the screen.

1. Functioning fighter jet
2. Shot down fighter jet
3. Mountains
4. Missiles
5. Sky
6. Score indicator
7. Start and pause button

5.2.1. Memory Budget:

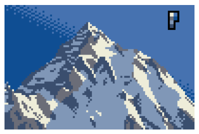
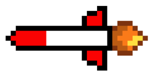
Element	Number of Sprites	Pixel Size	Size	Example
Mountains	2	70x85	5KB	
Sky	1	128x400	50KB	
Fighter Jet	2	40x80	4.7KB	
Missile	1	10x5	0.7KB	
Score	10	30x40	1.17KB	
Start/Pause	2	60x25	1.46KB	

Table 1: Graphics memory budget

6. AUDIO BLOCK

6.1. Audio Interface in DE1-SoC:

A high-definition 24-bit audio is provided by SSM2603 audio CODEC on the Cyclone V SocKit board. As shown in Fig 3., this chip enables microphone-in, line-in, and line-out ports. We will be using sound effects stored in the memory of the board and hence will not require the inputs for a microphone and a line-in. Simply playing our background music and sound effects suffices. SSM2603 supports a sampling rate adjustable from 8 kHz to 96 kHz. We plan on using 8kHz to conserve memory.

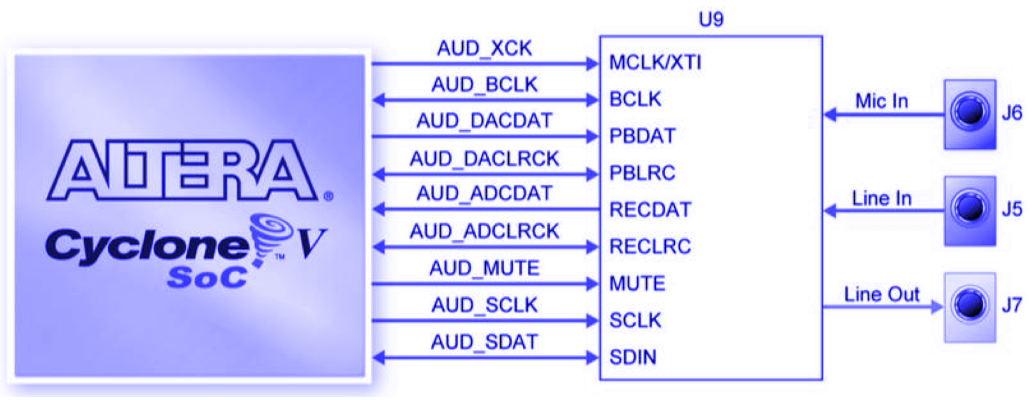
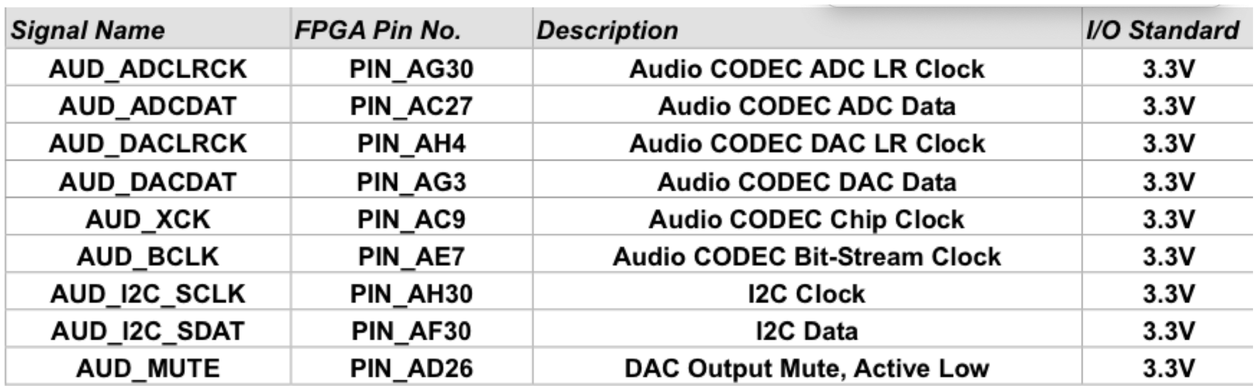


Fig 3. Interface between Audio CODEC and the FPGA



Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_AG30	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_AC27	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_AH4	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_AG3	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_AC9	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_AE7	Audio CODEC Bit-Stream Clock	3.3V
AUD_I2C_SCLK	PIN_AH30	I2C Clock	3.3V
AUD_I2C_SDAT	PIN_AF30	I2C Data	3.3V
AUD_MUTE	PIN_AD26	DAC Output Mute, Active Low	3.3V

Fig 4. Pin Assignment for Audio CODEC

6.2. Audio Block Interfacing:

6.2.1. I/O Protocol (I2C):

The SSM2603 (slave) is controlled via a serial I2C bus interface, which is connected to pins on the DE1-SoC (master). I2C is an IO protocol used to communicate between master and slave devices. There are read/write modes for master and slave devices.

- S/P: START and STOP symbol.
- SDIN(Serial data in): Transmission line for address, data and state symbol.
- SCLK(Serial clock): Global clock used for I2C buses
- ACK: Acknowledgement signal generated by receiver or slave

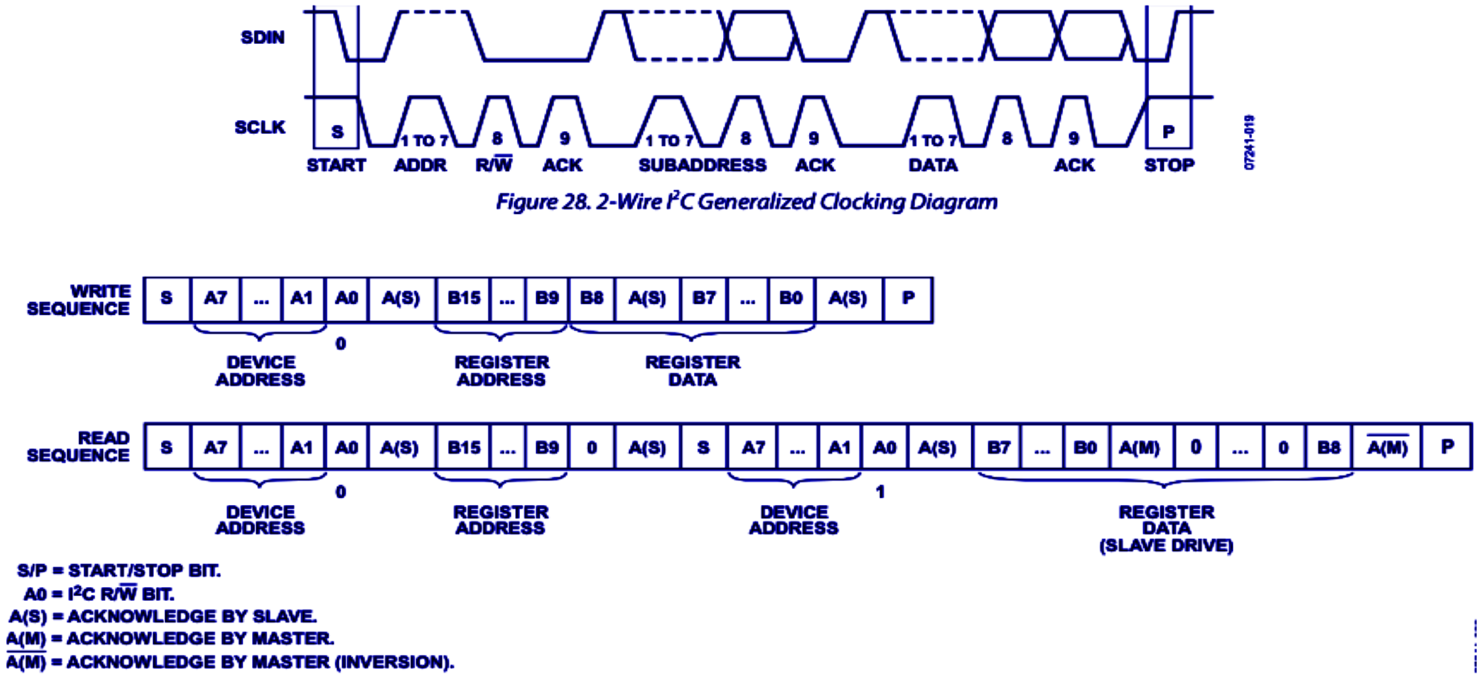

SDIN
SCLK
S
START
1 TO 7
ADDR
8
R/W̄
9
ACK
1 TO 7
SUBADDRESS
8
9
ACK
1 TO 7
DATA
8
9
ACK
P
STOP
07241-019
Figure 28. 2-Wire I²C Generalized Clocking Diagram
WRITE SEQUENCE
S | A7 | ... | A1 | A0 | A(S) | B15 | ... | B9 | B8 | A(S) | B7 | ... | B0 | A(S) | P
0
DEVICE ADDRESS
REGISTER ADDRESS
REGISTER DATA
READ SEQUENCE
S | A7 | ... | A1 | A0 | A(S) | B15 | ... | B9 | 0 | A(S) | S | A7 | ... | A1 | A0 | A(S) | B7 | ... | B0 | A(M) | 0 | ... | 0 | B8 | A(M) | P
0
1
DEVICE ADDRESS
REGISTER ADDRESS
DEVICE ADDRESS
REGISTER DATA (SLAVE DRIVE)
S/P = START/STOP BIT.
A0 = I²C R/W̄ BIT.
A(S) = ACKNOWLEDGE BY SLAVE.
A(M) = ACKNOWLEDGE BY MASTER.
A(M) = ACKNOWLEDGE BY MASTER (INVERSION).

Fig 5. I2C Write and Read Sequences

6.2.2. Audio Transmission Interface (I2S)

Inter-IC sound(I2S) is an electrical serial bus interface standard used for connecting digital audio devices together. We use I2S mode for our SM2603 with 32 bit ISA communicating with our speaker. There are also master and slave modes. Typically, the ADC IC is the master, and DAC is the slave. We use SM2603 in the master mode:

- WS(Word select): WS=0 left side soundtrack; WS=1 right soundtrack.
- SD(Serial Data):digital audio binary data.
- SCK(Serial clock): Global clock used for I2S buses.

The master has to send WS and SCK while the slave receives WS and SCK. In I2S, MSB is the first bit, and LSB is the last. Bit B15 to Bit B9 are the register map address, and Bit B8 to Bit B0 are data bits for the associated register map.

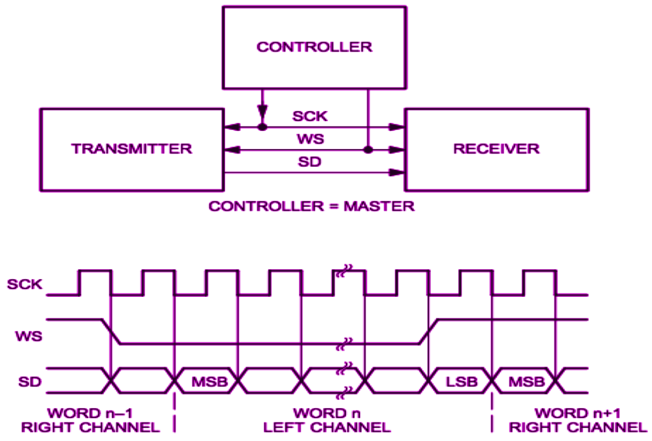


Fig 6. I2S Audio Input Mode

6.3. Audio Files:

We will use the following sound effects and background music during the game:

1. Score counting (0.5 sec)
2. Flight pitch (0.5 sec)
3. Hitting an obstacle (mountain/ missile) (0.5 sec)
4. Game over (1sec)
5. Background music: about 1 min and repeat until the game is over.

6.4. Memory Budget:

We assume that our sampling rate is 8kHz. An I2S data word consists of 16 bits. This means we use 16 bits to quantize one sampling point. According to these specifications, our sound effects are about 8 KB(0.5s) and 16KB(1s) and background music is around 1MB (the length of background music can be reduced to conserve memory if it becomes necessary). We are working with a 1GB DDR3 SRAM on-board. Coupled with the graphics memory budget we are currently well within the available memory.

7. MILESTONES

1. Implement hardware ports, controllers and drivers.
2. Implement graphics display with jet and obstacles. Test game screen with input control.
3. Implement game logic at low speed. Implement start/- pause/restart buttons.
4. Add difficulty levels and integrate audio output.