COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

CSEE 4840 Embedded Systems Design Project — Tetris

Zain Merchant (ztm2105)
Malik Hubbard (jmh2329)
Max Parson-Scherban (map2331)
Eva Gupta (eg3207)

Spring 2023

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

1 Introduction

Our Embedded Systems Design project is to develop the hardware and software for the classic game of Tetris on the DE1-SoC FPGA. Tetris is an originally Soviet-developed game where 7 4-block shapes, called Tetrominoes, fall into a playing field. The player controls the orientation of the Tetriminoes as they fall, through some mechanism of user input. Once a horizontal row of Tetromino blocks is complete, the line is deleted, higher blocks fall, and the user obtains a number of points for clearing the row. Upon a specified number of row clearings, the user "levels-up", and the rate of the descending Tetrominoes increases. The game ends when there is no additional space for a falling Tetromino to appear and descend. Our project will utilize the DE1-Soc FPGA, a USB NES controller, headphone, and a VGA monitor to recreate this classic game in SystemVerilog and the C programming language.

2 System Block Diagram

Figure 1 contains our System Block Diagram. It is expanded upon in later sections of this report.

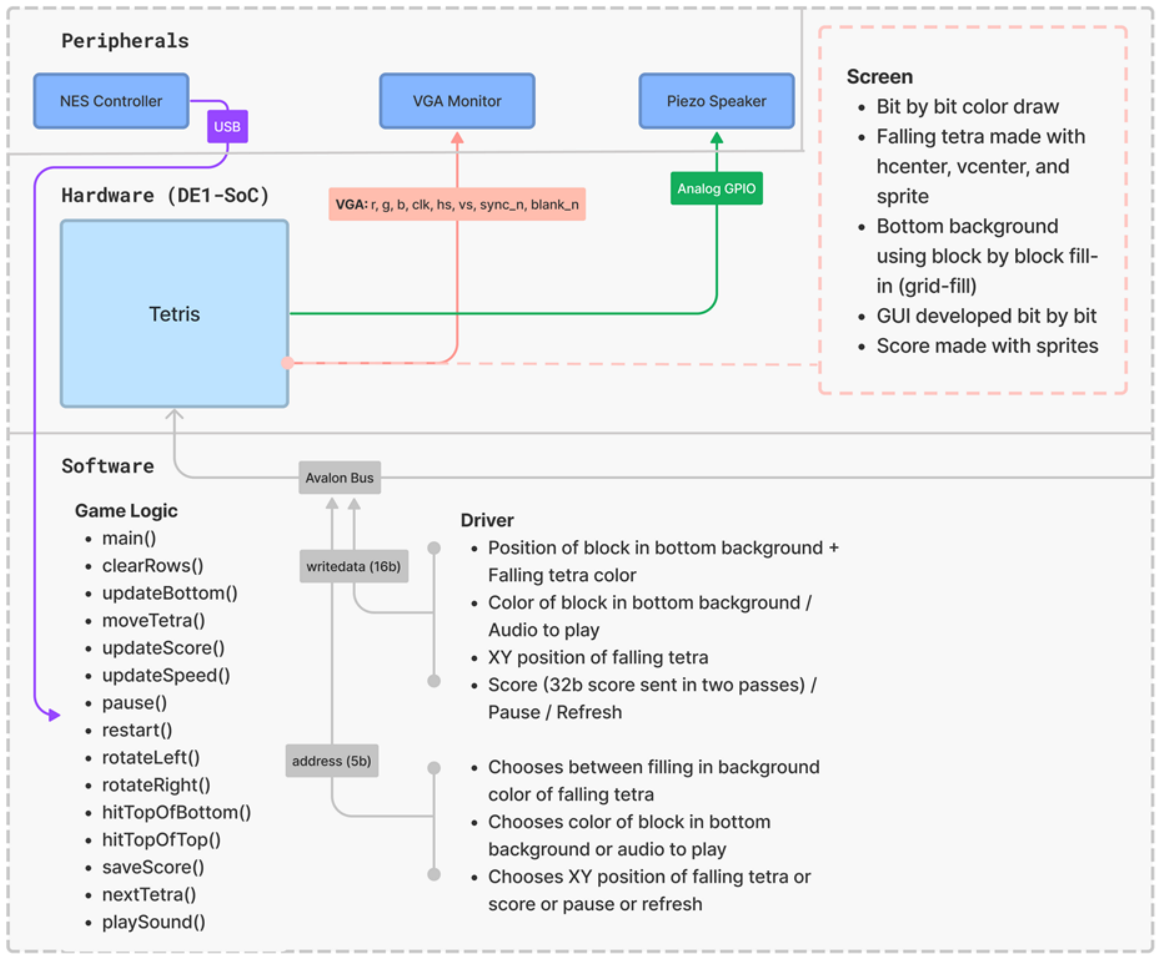

Peripherals
NES Controller
USB
VGA Monitor
Piezo Speaker
Hardware (DE1-SoC)
Tetris
VGA: r, g, b, clk, hs, vs, sync_n, blank_n
Analog GPIO
Screen
Bit by bit color draw
Falling tetra made with hcenter, vcenter, and sprite
Bottom background using block by block fill-in (grid-fill)
GUI developed bit by bit
Score made with sprites
Software
Avalon Bus
Game Logic
main()
clearRows()
updateBottom()
moveTetra()
updateScore()
updateSpeed()
pause()
restart()
rotateLeft()
rotateRight()
hitTopOfBottom()
hitTopOfTop()
saveScore()
nextTetra()
playSound()
writedata (16b)
address (5b)
Driver
Position of block in bottom background + Falling tetra color
Color of block in bottom background / Audio to play
XY position of falling tetra
Score (32b score sent in two passes) / Pause / Refresh
Chooses between filling in background color of falling tetra
Chooses color of block in bottom background or audio to play
Chooses XY position of falling tetra or score or pause or refresh

Figure 1: This is the block diagram of our Tetris system. A USB NES controller provides the input to our software drivers, which are responsible for our overall game logic. The drivers communicate with our VGA and GPIO logic on the FPGA to drive game output to a display.

3 Algorithms

3.1 Tetromino Rotation

We will use sprites to display each tetromino. A dual-ported memory unit will be utilized in hardware to save the sprites. Each sprite represents a tetromino and its orientation, thus for the 7 different tetrominoes each with 4 different orientations, there will be 28 different sprites to save. However, because the square tetromino is the same no matter how it is oriented and the 4x1 tetromino is the same for two of its states, we will actually only need to store 24 different sprites. When a falling tetromino is rotated we will simply change the sprite being shown on the VGA screen. A rotation of the sprite will occur using checks in software if it is allowed to do so by superimposing the currently rotating tetromino on a 2d array containing our game board state to match up if the resulting rotation would be allowed to fit within the confined of your current position and other tetrominos already placed on the board.

3.2 Bottom Background Addition

Our entire game space, a 10 by 20 block grid, will be represented in software as a two dimensional table. Each entry in the table is a block in the game space. Each entry will hold a numeric value that represents the color in that block. Every time the updateBottom function is called we will scan this array coloring each block in the game space the appropriate color based upon the array's value. Each block in the game space will be numbered from 0 to 255, which will correspond to the index of the 2D array. When a falling tetromino hits the top of the bottom background, based upon which sprite is being displayed the 2D array will be updated to color in the appropriate squares in the games space. The 2D array color values will then be pushed to the VGA display.

3.3 Block Fall/Move

The falling tetromino will exist independently of the bottom background. Similar to Lab 3, the center of each falling tetromino will exist in software and be communicated to the peripheral. Based upon the falling speed, the vertical center of the tetromino will be incrementing (moving down the screen). Similarly, when the appropriate keycode is received from the NES controller, the horizontal center of the block will be incremented or decremented appropriately while ensuring that it stays aligned with the columns of the game space.

3.4 Row Clear

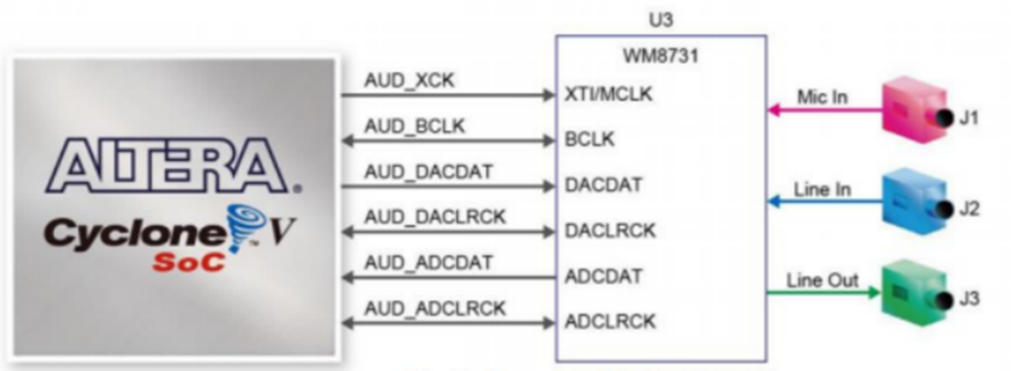
Clearing rows will take advantage of the software 2D array describing the bottom background. When a row is completed, it will first flash by updating the colors in the corresponding blocks in the 2D array. These changes will be pushed to the VGA display quickly to show the flash. To clear a row, we will update the 2D array then push the update to the VGA display. The 2D array will be updated as follows for row clears: the row that is cleared will take all the colors of the row above it and each row above the cleared row will take the colors of the row above it. This will represent the blocks falling after a row has been cleared. Every row below the cleared row will remain unchanged.

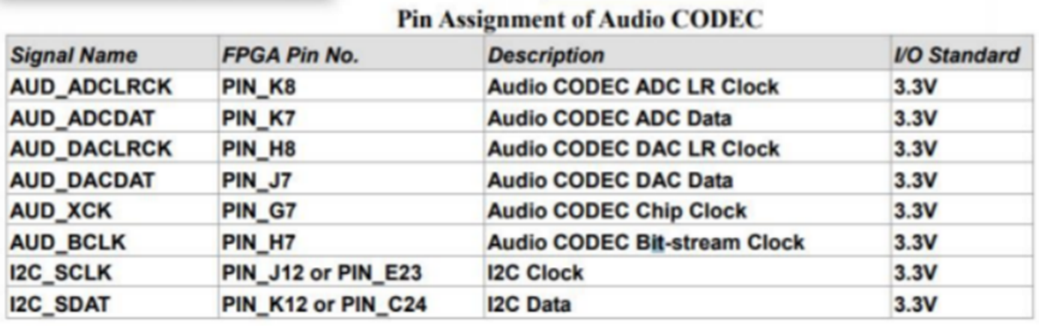
3.5 Next Tetromino

The next tetromino will be created using a random number generator. This generator will develop both the shape of the tetromino and the starting orientation of it. Based upon this randomly generated number, the next tetromino will be clearly defined.

4 Audio Generator

The DE1-SoC FPGA board is equipped with three audio jacks – a line out, line in, and a microphone jack. These jacks are linked to a Wolfson WM8731 audio CODEC chip, which generates ADCs and DACs to interface with analog jacks and the FPGA through a digital connection. To enable effective communication between Nios II processor and Audio CODEC, an audio core circuit is employed. This circuit simplifies the process of transmitting and receiving sound samples to and from the LINEOUT/MIC by incorporating two pairs of FIFOs, each containing 128 entries. These pairs correspond to the left and right channels and produce stereo sound. The circuit is below:

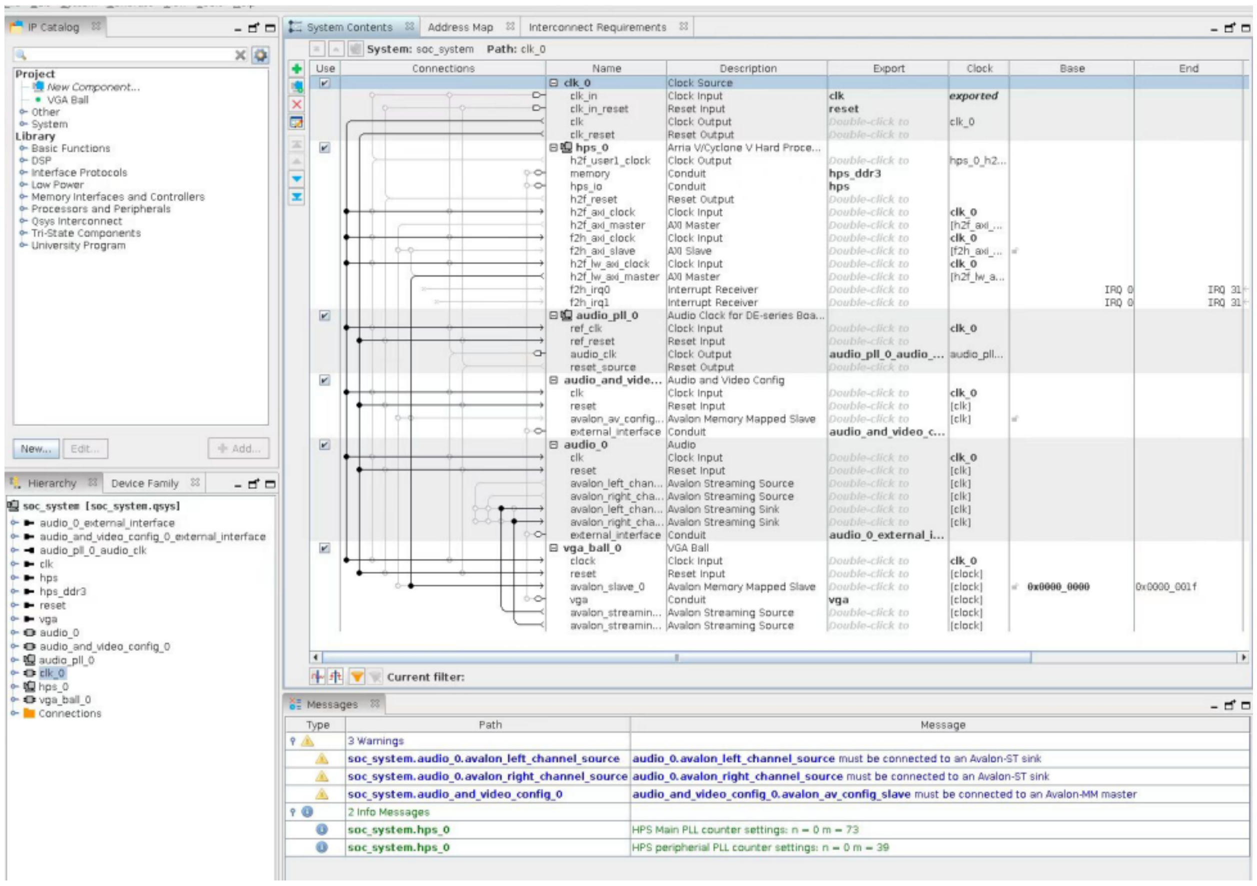



Pin Assignment of Audio CODEC

Signal Name	FPGA Pin No.	Description	I/O Standard
AUD_ADCLRCK	PIN_K8	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_K7	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_H8	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_J7	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_G7	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_H7	Audio CODEC Bit-stream Clock	3.3V
I2C_SCLK	PIN_J12 or PIN_E23	I2C Clock	3.3V
I2C_SDAT	PIN_K12 or PIN_C24	I2C Data	3.3V

Figure2: Schematic Diagram

For the sound implementation, the team first downloaded tetris theme song from the internet, then sampled it at 8KHz through an online converter. Then we convert this file to 16 bit mono PCM.wav as WM8731 is left justified mode, meaning, the information at the left side is same as the right side of the channel. Once this was done, the team made sure that the bit rate was 128Kbps. Then convert this .wav file to a .mif file. But when we were implementing the sound using a .mif file there were some errors so we decided to convert this .mif file to a .hex to have a proper implementation.



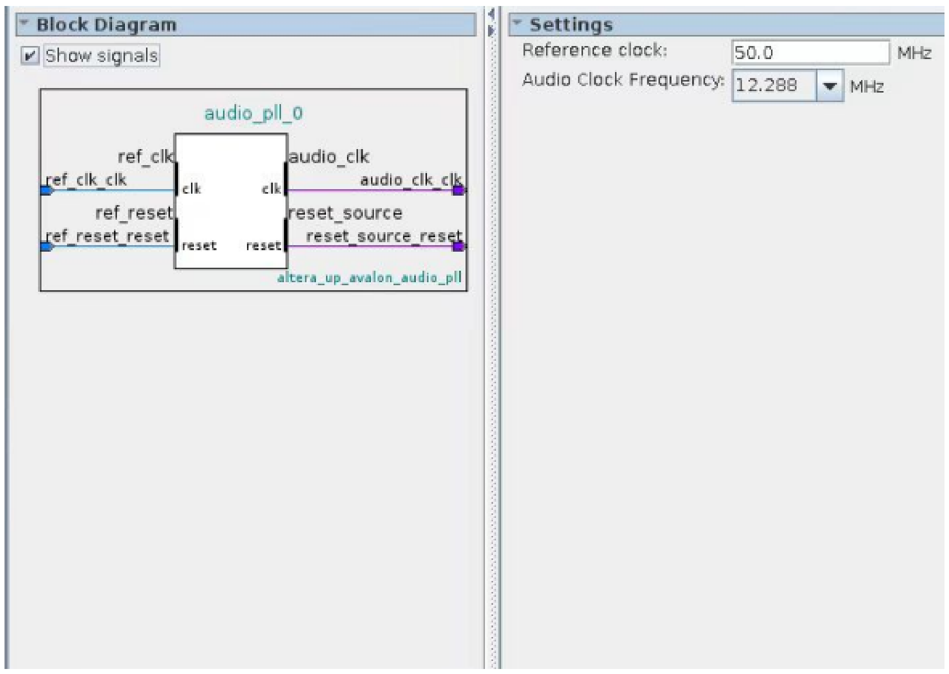
System: soc_system Path: clk_0

Name	Description	Export	Clock	Base	End
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			
hps_0	Arria V/Cyclone V Hard Proce...				
h2f_user1_clock	Clock Output	Double-click to	hps_0_h2...		
memory	Conduit	hps_ddr3			
hps_io	Conduit	hps			
h2f_reset	Reset Output	Double-click to			
h2f_axi_clock	Clock Input	Double-click to	clk_0		
h2f_axi_master	AXI Master	Double-click to	[h2f_axi_...		
f2h_axi_clock	Clock Input	Double-click to	clk_0		
f2h_axi_slave	AXI Slave	Double-click to	[f2h_axi_...		
h2f_lw_axi_clock	Clock Input	Double-click to	clk_0		
h2f_lw_axi_master	AXI Master	Double-click to	[h2f_lw_a...		
f2h_irq0	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31
f2h_irq1	Interrupt Receiver	Double-click to		IRQ 0	IRQ 31
audio_pll_0	Audio Clock for DE-series Boa...				
ref_clk	Clock Input	Double-click to	clk_0		
ref_reset	Reset Input	Double-click to			
audio_clk	Clock Output	audio_pll_0_audio_...	audio_pll...		
reset_source	Reset Output	Double-click to			
audio_and_vide...	Audio and Video Config				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
avalon_av_config...	Avalon Memory Mapped Slave	Double-click to	[clk]		
external_interface	Conduit	audio_and_video_c...			
audio_0	Audio				
clk	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clk]		
avalon_left_chan...	Avalon Streaming Source	Double-click to	[clk]		
avalon_right_cha...	Avalon Streaming Source	Double-click to	[clk]		
avalon_left_chan...	Avalon Streaming Sink	Double-click to	[clk]		
avalon_right_cha...	Avalon Streaming Sink	Double-click to	[clk]		
external_interface	Conduit	audio_0_external_i...			
vga_ball_0	VGA Ball				
clock	Clock Input	Double-click to	clk_0		
reset	Reset Input	Double-click to	[clock]		
avalon_slave_0	Avalon Memory Mapped Slave	Double-click to	[clock]	0x0000_0000	0x0000_001f
vga	Conduit	vga	[clock]		
avalon_streamin...	Avalon Streaming Source	Double-click to	[clock]		
avalon_streamin...	Avalon Streaming Source	Double-click to	[clock]		

Messages

Type	Path	Message
	3 Warnings	
	soc_system.audio_0.avalon_left_channel_source	audio_0.avalon_left_channel_source must be connected to an Avalon-ST sink
	soc_system.audio_0.avalon_right_channel_source	audio_0.avalon_right_channel_source must be connected to an Avalon-ST sink
	soc_system.audio_and_video_config_0	audio_and_video_config_0.avalon_av_config_slave must be connected to an Avalon-MM master
	2 Info Messages	
	soc_system.hps_0	HPS Main PLL counter settings: n = 0 m = 73
	soc_system.hps_0	HPS peripherial PLL counter settings: n = 0 m = 39

Figure3: Final Qsys Connections

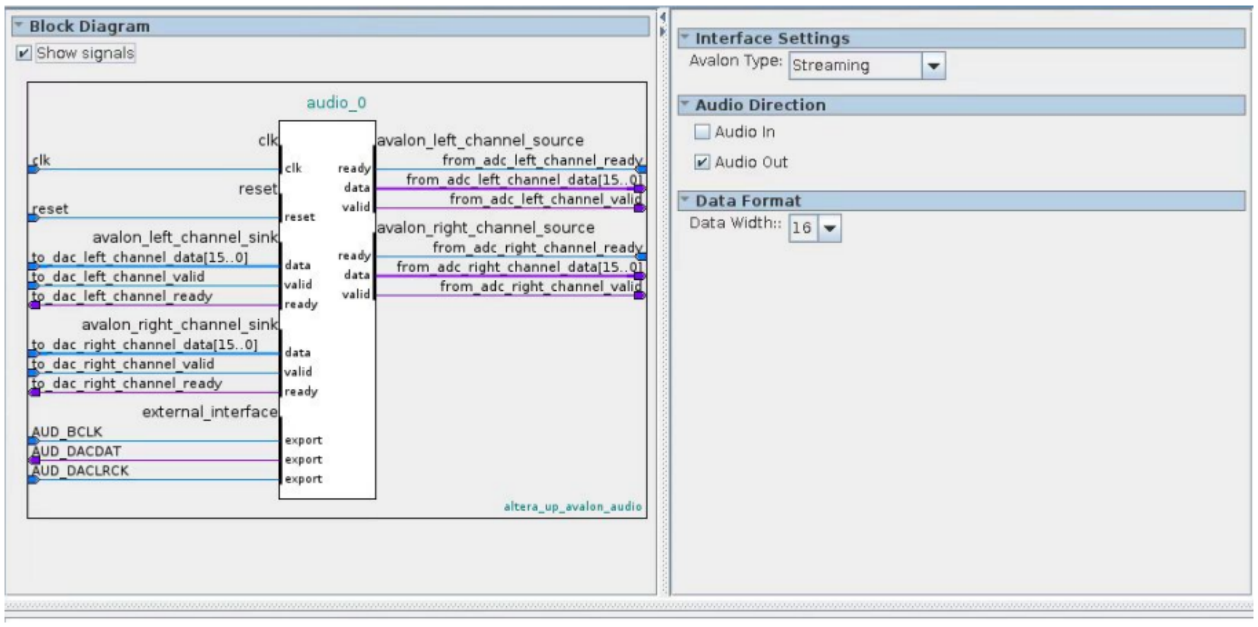


Settings

Reference clock: 50.0 MHz

Audio Clock Frequency: 12.288 MHz

Figure4: audio_pll_0


Block Diagram
Show signals
audio_0
clk
clk
clk
reset
reset
reset
avalon_left_channel_sink
to_dac_left_channel_data[15..0]
to_dac_left_channel_valid
to_dac_left_channel_ready
data
valid
ready
avalon_right_channel_sink
to_dac_right_channel_data[15..0]
to_dac_right_channel_valid
to_dac_right_channel_ready
data
valid
ready
external_interface
AUD_BCLK
AUD_DACDAT
AUD_DACLRCK
export
export
export
avalon_left_channel_source
from_adc_left_channel_ready
from_adc_left_channel_data[15..0]
from_adc_left_channel_valid
ready
data
valid
avalon_right_channel_source
from_adc_right_channel_ready
from_adc_right_channel_data[15..0]
from_adc_right_channel_valid
ready
data
valid
altera_up_avalon_audio

Interface Settings

Avalon Type: Streaming

Audio Direction

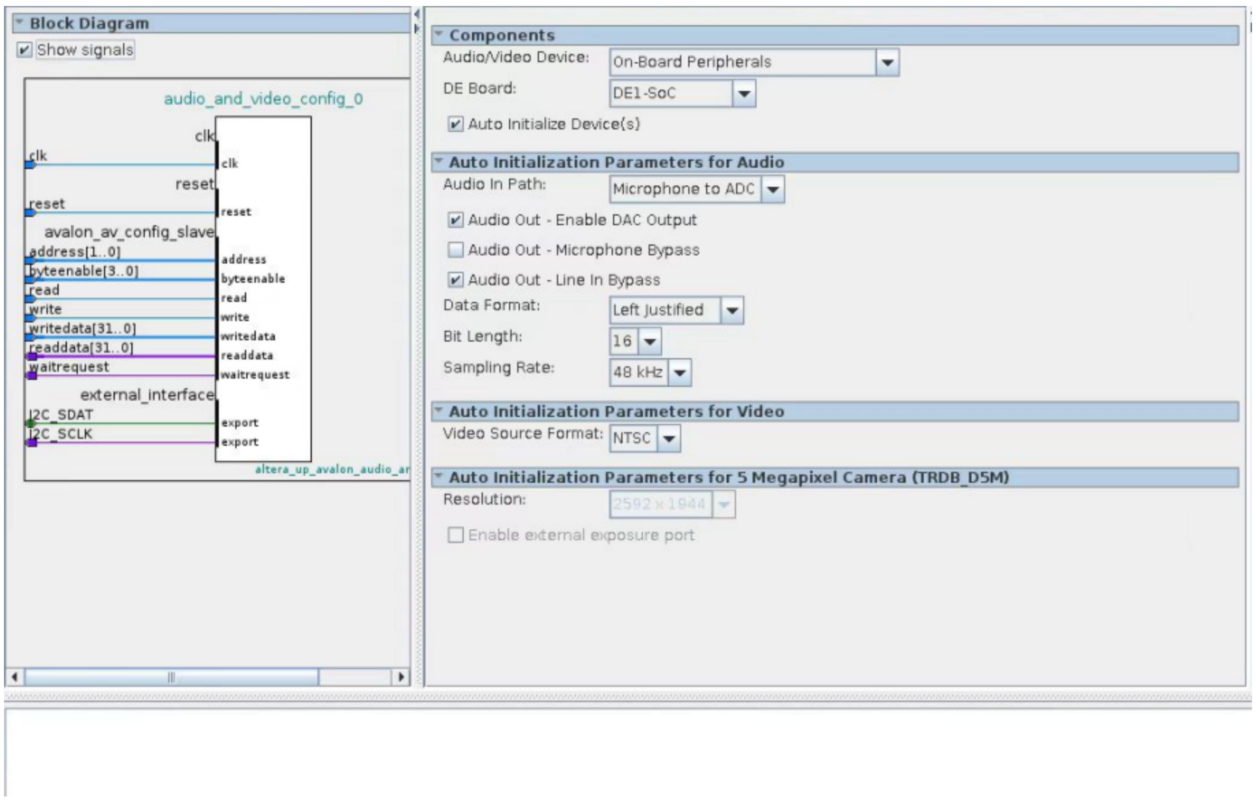
☐ Audio In

☑ Audio Out

Data Format

Data Width:: 16

Figure5: audio_0



Components
Audio/Video Device: On-Board Peripherals
DE Board: DE1-SoC
[✓] Auto Initialize Device(s)

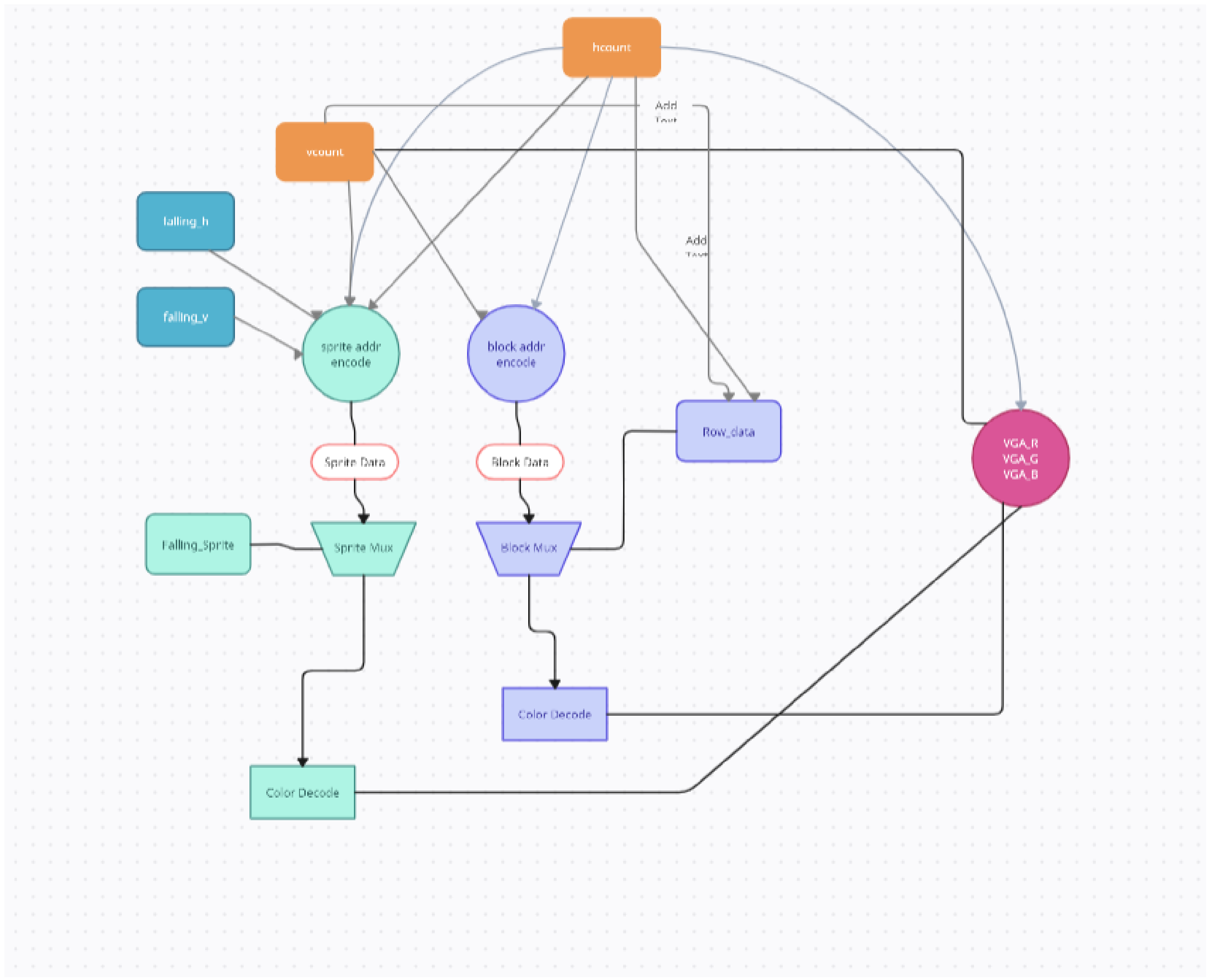
Auto Initialization Parameters for Audio
Audio In Path: Microphone to ADC
[✓] Audio Out - Enable DAC Output
[ ] Audio Out - Microphone Bypass
[✓] Audio Out - Line In Bypass
Data Format: Left Justified
Bit Length: 16
Sampling Rate: 48 kHz

Auto Initialization Parameters for Video
Video Source Format: NTSC

Auto Initialization Parameters for 5 Megapixel Camera (TRDB_D5M)
Resolution: 2592 x 1944
[ ] Enable external exposure port

Figure6: audio_and_video_config_0

5 Hardware Design



5.1 Graphics

5.2 Background

The background of the game space was maintained in hardware as a 2D array of characters, with each character communicating the color of that block. Each color could be blue, white, or red, each with a unique sprite. In a loop, the software would send a 32-bit integer whose top 20 bits encoded the color of each of the 10 blocks on that row. Once the data was written to the appropriate register for a certain row, the the register was demultiplexed to determine what color each block should be. Then, for each pixel in a particular block rangethe hardware determines which block it is in, using hcount to determine the block and hcount and vcount to determine where the sprite pixel lies. The hardware then consults the block CLUT for that pixel and writes the appropriate value to the screen.

5.3 Block

The falling block is communicated through software using an integer and is saved into the falling_reg register in hardware. The bottom 8 bits of the falling_reg register are used to determine 1 of the 25 tetromino sprites by selecting which tetromino sprite memory data signal is fed into the tetromino CLUT. As the tetromino falls, the falling_h and falling_v registers are updated. These values are used to determine where the tetromino is on the screen in hardware and draw the sprite around the tetromino falling values top-left corner.

5.4 Numbers

Numbers are used to communicate to the player the number of lines they have cleared. This value is updated by communicated through the background register. Based upon the value in this register, the sprites for the numbers are updated.

6 Software Design

6.1 User Input

User input is done through a USB NES controller, similar to how it was implemented in Lab 2. The user can use pointer keys to select tetromino movement in all directions except up. Multiple characters cannot be input at once on the keys, and they key selection triggers the game logic functions that determine if a move is possible or not before acting on it. The A and B keys rotate the block clockwise or counterclockwise if possible and stop at rotating a specific direction if it cannot pass that rotation state. We used libusb for interacting with the USB peripheral.

6.2 Game Logic

The software is designed to have two threads, one for the input from the controller and the other for handling game logic and pushing changes to hardware. Each loop is one frame of the game, with a fixed delay between each one. The input keys data gets sent to the screen thread, where its motion is processed. The game contains an internal ascii representation of what is going on which greatly aided in debugging. Each move possibility is superimposed on this underlying game state to see if it is possible before making it. We add in the additional movement for the gravity of the game. Next we check if we're in a game over state and handle that accordingly. We then check for any complete rows and shift down the game board accordingly to tackle that. Lastly we update with a new tetromino and restart the loop. All of these functions correspond to the ascii representation and update the screen.

7 Hardware and Software Interface

State of Unmoving Tetrominos

Background Row 0 - Address 0 - 30 bit int that will describe the color (7 possible) of the 10 blocks in row 0 (the bottom row).

Background Row 1 - Address 1 - 30 bit int that will describe the color (7 possible) of the 10 blocks in row 1 (second from the bottom).

...

Background Row 19 - Address 19 - 30 bit int that will describe the color (7 possible) of the 10 blocks in row 19 (the top row).

Background Next - 3 bit int that will select which tetromino sprite will be displayed as next

State of Falling Tetromino

Falling Sprite - Address 22 - 3 bit int that will select the sprite for the falling tetromino.

Falling Vertical - Address 20 - 10 bit int that will set the vertical position for the falling tetromino.

Falling Horizontal - Address 21 - 10 bit int that will set the horizontal position for the falling tetromino. This is just selecting one of 10 columns.

Falling Ori - Address 28 - 2 bit int that will control the orientation of the falling tetromino. The default orientation is 00, but when the block is turned, this value will be updated.

State of Score

Current Score - Address 24 - 32 bit int that will communicate the player's score.

High Score - Address 25 - 32 bit int that will communicate the high score for the Tetris session.

State of Music

Music Enable - Address 27 - 1 bit that will signal for the sound indicated by Music Sound to be played.

Music Sound - Address 26 - 4 bits that will select which sound to play.

8 Problems Faced and Individual contributions

8.1) Hardware

Max and Malik worked on the hardware part of the project and faced challenges with the falling terominos (falling_h and falling_v). One other issue they faced was the multiplexing of the sprite during the implementation.

8.2) Software

Zain was working with the game logic and setting up the software part of this project. And faced some challenges in software that were faced include the initial representation of the game state in ASCII and the implementation of the logic of the aforementioned rotation and movement functions. Additionally, we had some issues with the controller input handling being delayed from when it was actually expected. Zain also had challenges interacting with the lower level hardware and helped debugging that with Max and Malik.

8.3) Audio

Eva faced the challenge of setting up Qsys initially as the documentation online was not clear enough. She then ended up referring to previous year's projects for setting up Qsys and converting the audio files. Using Audacity software was not possible as it couldn't be opened so she used an online converter to convert .mp3 file to .wav file and then wrote a python script to convert this .wav file to .mif file. But during the compilation process the .mif file was throwing an error of syntax error. Hence, we decided to convert the .mif file to .hex file.