ECE 385

Spring 2023

Final Project

Tetris

Dylan Bautista, Matthew Wei

Lab Section NL/ May 5 11:00 AM

Nick Lu

Final Project

Tetris

I. Introduction/Purpose of Circuit:

Our final project circuit is supposed to replicate the classic NES game of Tetris through our SOC design and software code. Using a USB keyboard as input and a VGA display as output, the implementation contains all the basic functions of stacking blocks, clearing rows, score counting, and game ending. Additionally, a starting and ending screen, color display, and holding a block were added to the gameplay. As usual, a NIOS II CPU is used alongside a RAM component for on-chip storage, with most of the screen logic being written in C software with VGA controls based in the VGA text AVL interface module.

II. Written description of Circuit

In terms of emulated hardware, the Tetris circuit is relatively similar to the previous 7.2 and 6.2 labs where we implemented VGA sprite drawing and USB interfacing. The VGA_text_mode component was added to our platform designer layout along with the accompanying vga_text_avl_interface.sv file. The .sv file allows us to interact with the signals from the NIOS II processor via reading and writing the Avalon slave signals along with the color_mapper/ VGA_controller signals. We instantiated an on-chip memory mega function, creating a RAM module which can be instantiated within the VGA_text_interface file to hold up to 2048 words of memory. Additionally, there is a 7-color palette register where the color information is stored. In this file, we determine if the DrawX/DrawY location is within areas such as the grid space, a Tetris block, the border, the scoring location, or just blank space. We then correctly color the space using the palette register, or alternatively access font_rom or paint

the entire screen according to the start image. The start image is created utilizing the provided helper tools for converting images to external RAM and color palette.

Software Implementation

The bulk of the implementation lies in C code, where we alter the data stored in the RAM locations which are accessed by the VGA interface module. In a TEXT_VGA_STRUCT that we create in a Tetris header file, we allocate 5 sets of 20 words of memory. Firstly, is the VRAM which holds the 20 rows of bits of if a space is occupied, then currBlock which just has the current block as ones, then colors which holds the rows of color indexes, then rotBlock which just has the current block's rightward rotation as ones, then color_block which just has the current block's color indexes. Additionally, there are separate variables in memory such as game_start, game_over, the decimal score, and the 8 digits of the score to communicate this information to the VGA interface.

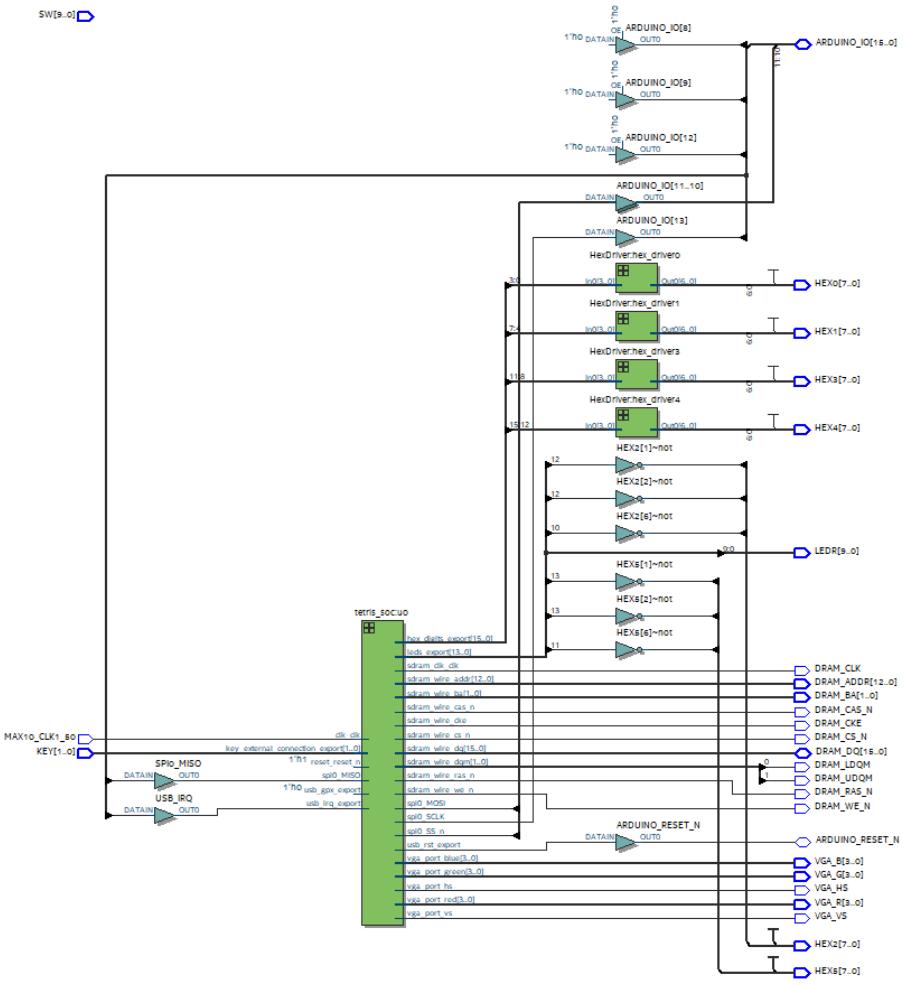
The main function utilizes the 6.2 USB interfacing C code, using a USB driver on the NIOSII in order to communicate with the MAX3421 via the SPI peripheral. From this, we receive the keycodes sent in at any given moment and send them in as arguments to the move function in Tetris.c. The main function also calls the startGame function and consistently calls the move down function.

The startGame function initializes global variables, the score, and memory contents, then calls gen_block. This generates a random block at the top, if possible, from a stored piece library. From the move function, depending on what the keycode is, we call either moveLeft, moveRight, moveDown, or rotate. These all use the current state of VRAM, currBlock, and possibly rotBlock in order to determine if the boundaries and previously placed blocks allow for the movement. If

it is allowed, the contents of memory are changed, and a function is called to update the memory of both colors and color_block. Each time moveDown is called, if it can't be moved, we check if any lines are full and generate a new block. If any lines are full, we remove them, shifting everything down, and call a function to update the score memory data. If a block cannot be generated, the endgame function is called which halts player activities and signals to the VGA interface to flash "Game Over".

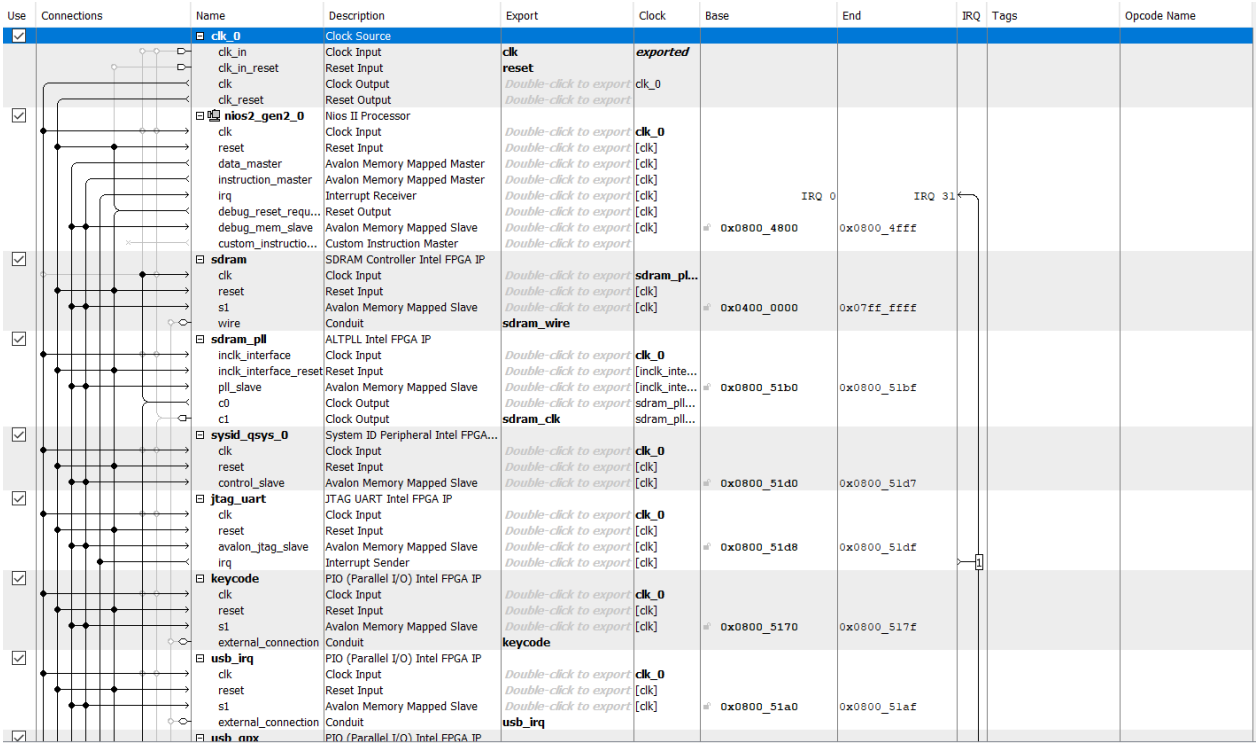
III. Block Diagram

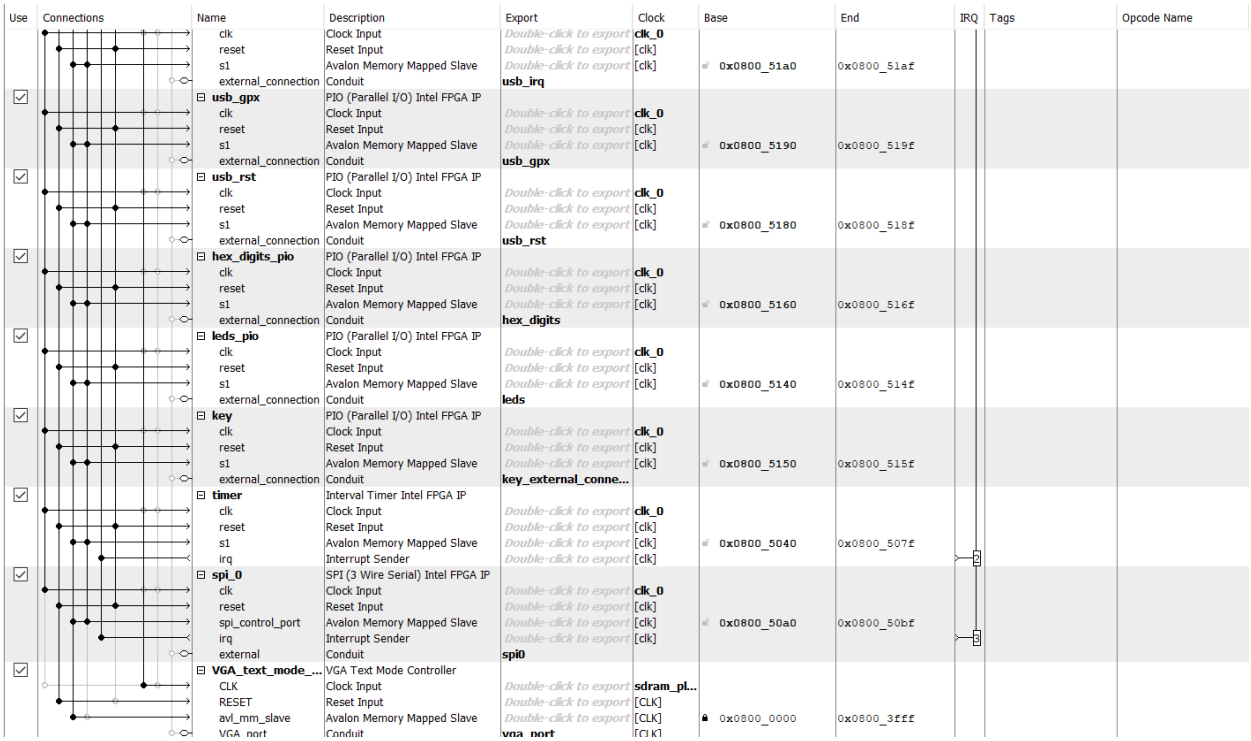
Top Level Diagram



Block Diagram

Platform Designer View





Platform Designer

IV. Modules and Blocks

SystemVerilog Modules

font_rom.sv

Inputs: [10:0] addr

Outputs: [7:0] data

Description: Module that holds information about each character and how each pixel is represented for each character.

Purpose: Allows retrieval of character structure information (which pixel should be colored what for each character ie. foreground or background).

HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: Module that holds information about the Hex LED displays on the FPGA board.

Purpose: Used so the correct output can be displayed onto the FPGA HEX displays.

tetris.sv

Inputs: MAX10_CLK1_50, [1:0] KEY, [9:0] SW

Outputs: [9:0] LEDR, [7:0] HEX0, [7:0] HEX1, HEX2, HEX3, HEX4, HEX5, DRAM_CLK, DRAM_CKE, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3:0] VGA_R, VGA_G, VGA_B

Description: Instantiates all the modules required for our design.

Purpose: Top-level for our design.

tetris_soc.qip

Description: Module that holds information about the hardware created in Platform Designer.

Purpose: Used so outside hardware can be interacted with from user.

ram.v

Inputs: [10:0] address_a, address_b, [3:0] byteena_aclock, [31:0] data_a, data_b, rden_a, rden_b, wren_a, wren_b

Outputs: [31:0] q_a, q_b

Description: Module created by Quartus GUI Megafunction to hold information about character pixel data and color data. Two-port RAM with inputs listed above.

Purpose: Used as the RAM to hold the contents of every word (replacement of the 601 registers in 7.1 except now has 1200)

start_example.sv

Inputs: vga_clk, blank, [9:0] DrawX, DrawY

Outputs: [3:0] red, green, blue

Description: Calculates the red, green, and blue values for the current pixel. Instantiates rom and palette.

Purpose: Instantiates the rom and palette for the start screen and calculates the RGB values for the current pixel.

start_palette.sv

Inputs: [3:0] index

Outputs: [3:0] red, green, blue

Description: Gets the red, green, and blue values for a specific color by indexing into registers.

Purpose: Gets the correct color from the palette registers.

start_rom.sv

Inputs: clock, [16:0] address

Outputs: [3:0] q

Description: Holds information about what color each pixel should be for the graphic (start screen).

Purpose: Holds information about the start screen data.

VGA_controller.sv

Inputs: Clk, Reset,

Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, DrawY

Description: Module that sets the pixel value onto the screen using vertical and horizontal syncs.

Purpose: Used to draw or set pixel values on the 640x480 VGA screen.

vga_text_avl_interface.sv

Inputs: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, [11:0] AVL_ADDR, [31:0] AVL_WRITEDATA,

Outputs: [31:0] AVL_READDATA, [3:0] red, green, blue, hs, vs

Description: Instantiates RAM, font data, and VGA controller and gets information about from RAM about what to print on the screen as well as other additional data.

Purpose: Accesses RAM data from Avalon bus to properly display and print out the correct objects' specific spots and states of the Tetris game.

System Blocks

clk_0

Function: Main 50 MHz clock used for the entire design and all components

nios2_gen2_0

Function: NIOS II/e processor block. It sends instructions and data to other blocks.

sdram

Function: Off-chip 512 MB memory utilizing sdram_pll as a clock to read and write operations correctly.

sdram_pll

Function: Clock delayed 1ns from main clock to accurately have sdram work.

sysid_qsys_0

Function: Ensures no mismatch or incompatibilities between hardware and software incorporation.

jtag_uart

Function: Allows software to communicate with FPGA by outputting onto terminal.

hex_digits_pio

Function: 16-bit output. Hex_digits_pio is the Hexdigit displays on the FPGA board.

keycode

Function: 8-bit input value that holds which keyboard key was pressed.

usb_irq/usb_gpx/usb_rst

Function: 1-bit PIOs that allows keyboard to communicate with FPGA.

key

Function: 2-bit input value. Key is the of the 2 buttons on the FPGA board (run and reset).

timer

Function: Clock to keep track of how much time has passed.

spi_0

Function: Allows data transfer between master and slave components.

vga_text_mode_controller_0

Function: Allows VGA signals to be linked from the Avalon bus to the VGA monitor.

V. Conclusion

Overall, we were able to get a successful implementation of Tetris and got full functionality points for our project. We were able to secure an 8 on difficulty as well with the inclusion of colored blocks, on-screen text, hold piece, and a title screen. Overall, the final project was not too bad. The hardest part about it was the first week when we were trying to figure out how to implement Tetris and which approach to use. Once we got that figured out, it was pretty simple to implement what functions we needed for our functionality. Most of our code was done in C as well, so a lot of time was saved compiling and debugging in SystemVerilog.

For bugs, there were a lot that came up that we had to debug. Such bugs included were wrong pixel drawing, wrong font drawing, wrong rotations, hold piece glitch, incorrect down movement, incorrect sideways movement, and incorrect color printing. Fixing these bugs were either a simple change in code or an addition of a new block of code/boundary checking. Pixel drawing and font drawing bugs were caused by a wrong pixel boundary in SystemVerilog and wrong access to VRAM memory. Rotation, movement, and color printing were usually the result of incorrect setting of VRAM memory or a missing case check (top row, right/left-most column, bottom row, etc.). The hold piece bug had one of the pieces using the wrong identifier value, so it would return an incorrect piece when trying to swap to it. Finally, the down movement bug was

in main.c where the block would move down faster if a keyboard input was detected. This was fixed by adding some buffers to prevent the keyboard from speeding up the movement.

Post-Lab Question

	Tetris
LUT	19,106
DSP	0
Memory (BRAM)	119,808
Flip-Flip	2,826
Frequency (MHz)	140.53
Static Power (mW)	96.18
Dynamic Power (mW)	0.75
Total Power (mW)	106.24