Project Report: Tetris

Prof. Stephen Edwards

Spring 2020

Arsalaan Ansari (aaa2325)

Kevin Rayfeng Li (krl2134)

Sooyeon Jo (sj2801)

Josh Learn (jrl2196)

Table of Contents

Introduction

The purpose of this project was to build a Tetris video game system using System Verilog and C language on a FPGA board. Our Tetris game is a single player game where the computer randomly generates tetromino blocks (in the shapes of O, J, L, Z, S, I) that the user can rotate using their game controller. Tetrominoes can be stacked to create lines to be cleared by the computer and be counted as points that will be tracked. Once a tetromino passes the boundary of the screen the user will lose.

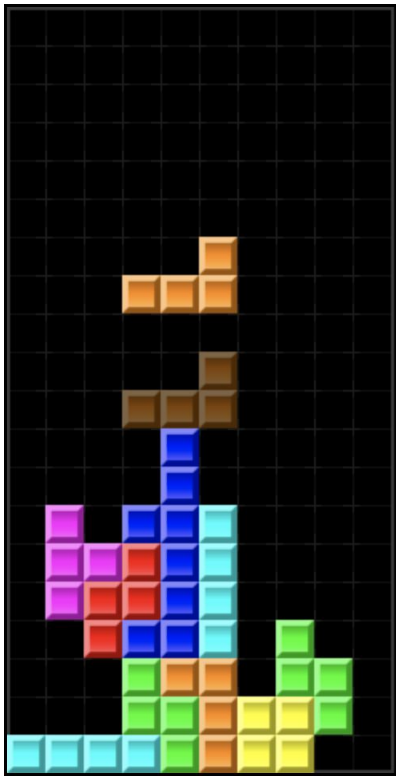

Fig 1: Screenshot from an online implementation of Tetris

User input is taken from key inputs from a game controller, and the Tetris sprite based output is displayed using a VGA display. The System Verilog code creates the sprite-based imagery for the VGA display and communicates with the C language game logic to change what is displayed. Additionally, the System Verilog code generates accompanying audio that will supplement the game in the form of sound effects. The C game logic generates random tetromino blocks to drop, translate key inputs to rotation of blocks, detect and clear lines, determine what sound effects to be played, keep track of the score, and determine when the game has ended.

Architecture

The figures below show the proposed and actual architecture for our project.

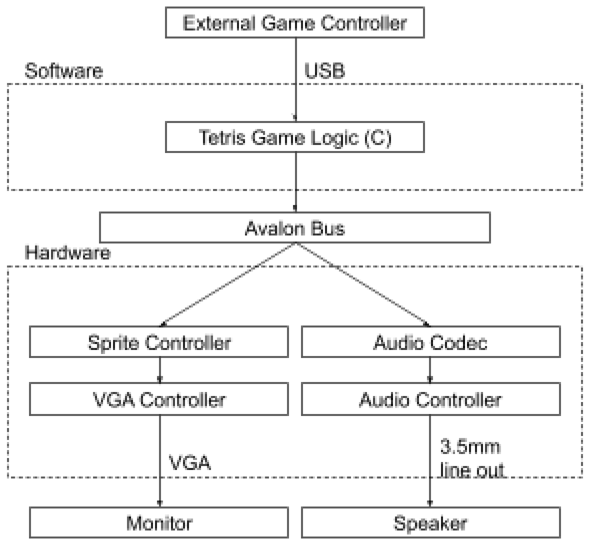

Fig 2.1: Proposed architecture

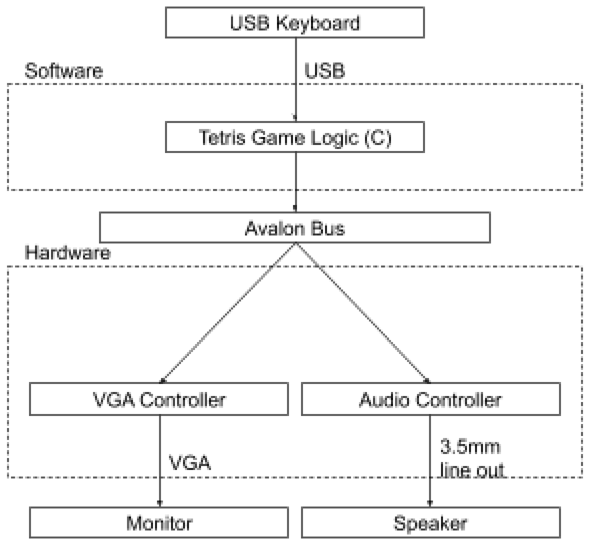

Fig 2.2: Actual architecture

As you can see, we simplified the design greatly for the final implementation. We discuss challenges in a later section in this report.

Hardware Implementation

VGA Block

We intended our Tetris game to have 3 layers of graphics. The bottom layer would contain the background of the game. The middle level would contain the falling and resting tetrominos. The surface layer would show the counter that keeps track of the score and a “Game Over” overlay.

In our final implementation, we simply had squares that were "drawn over" the VGA outputs, overriding a background layer. This was a prototyping decision to make it easy to implement sprites later, but we never got to that stage.

Each of the tetrominos will correspond to four identical sprites, which we simplified to squares in our final design, which are the component blocks that make it up. We will have signals coming from software that we map to each new tetromino block. Our hardware side simply receives the indices of blocks it has to render on each tick of the game. This means that enough clock cycles must pass between each in-game tick for the hardware to receive the data (1 or 0) for all of the possible block locations. This is no issue in our prototype as we pass all the hardware data in a single clock cycle, and only refresh the screen at max once every 500μs, far less frequent than the FPGA clock cycle.

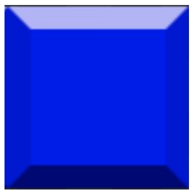

Fig 3.1: A proposed sprite for one tetromino block. This has been simplified to just a colored square in our prototype.

Given that we weren't able to implement the score counter we only had two layers of graphics, the background, and the tetromino blocks. The blocks were not generated with sprites, instead the SystemVerilog code read a 32 bit bitmap of data from writedata every in-game tick. The SystemVerilog code contains conditional statements that activate blocks of the screen based on the bits found in the 32 bit bitmap. The hardware is not aware of the game logic, it only exists to render blocks on the screen based on what the software shows.

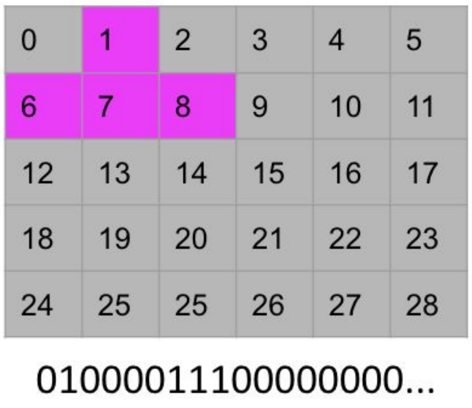

Figure 3.2: An example of how a tetromino might be encoded into a 32bit data packet for the ioctl

As can be seen above, the bits are set according to their position in the grid. This was done to theoretically make it easy to align sprites to the grid later, which we were unable to get to. One drawback of this is that how many blocks we could display was bottlenecked by the size of the memory region writable by I/O (32 bits in case of our board). Another drawback was that we were initially intending to have smooth movement of the blocks, but this data format didn't support that in our display.

Wolfson WM8731

For the audio functionality, we will be using the Wolfson WM8731 CODEC on the DE1-SoC board, which offers 24-bit audio. We implemented audio by creating a music module that generates tones. We weren’t able to test functionality due to qsys errors.

DE1-SoC I^{2}C Multiplexer

We will be using the I^{2}C multiplexer to determine if the input to the I^{2}C bus comes from the HPS or the FPGA part. The FPGA part will handle the I^{2}C configuration.

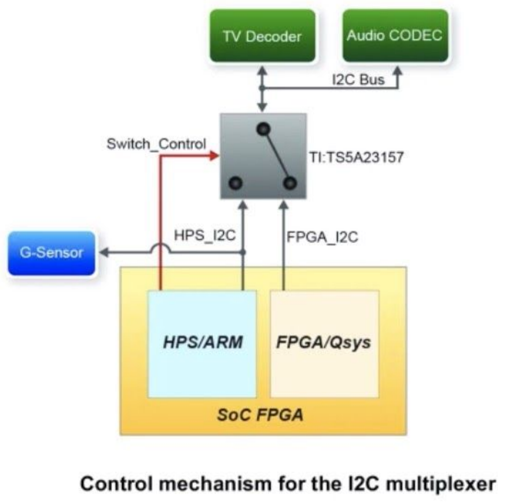

Fig. 4: I^2C Multiplexer

Software Implementation

Game Logic

All game logic is controlled through C code that communicates with the System Verilog program.

Tetromino control: The movement and rotation of the Tetrominos were intended to be controlled by a game controller. Tetrominoes can be moved left and right using inputs from the keyboard. Rotation of the tetromino is to be done with the space key, and is yet to be implemented.

Tetromino generation: Tetromino generation is handled through a random number generator algorithm that can choose between the 6 shapes. Another random number generator will determine the initial rotation of the tetromino.

Finally, a third random number generator will determine what X coordinate the piece will fall from (keeping in mind that the piece generated from before fits within the bound of the screen at that coordinate).

Line Clearing: (To be implemented) The software will keep track of every X coordinate pixel at each Y coordinate row. When all pixels in a row are detected to be filled the line will clear, and the software will tell the hardware to shift the display accordingly.

Game Over: The software will detect if the most recent tetromino placed has any y coordinates greater than the bounds of the screen. If so the game will end and a Game Over overlay should appear.

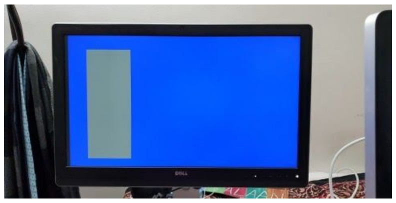
Fig 5.1: Initial game state

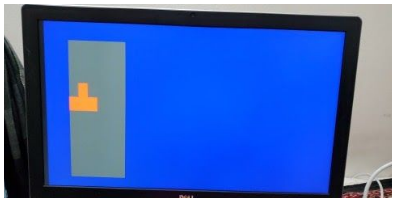
Fig 5.2: First piece generated

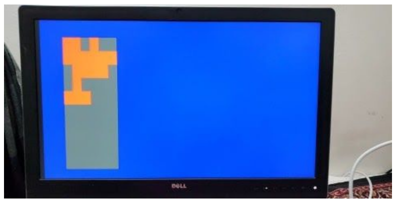
Fig 5.3: Finished game state

Audio

We planned for our game to have sounds in the following times:

1. When the game starts
2. While the piece rotates
3. When the player clears the row

4. When the player finishes the game

We intended to synthesize the sound to save on game memory, and modulate it to create the sound effects necessary. Because of time and hardware constraints, our implementation was primitive, simply playing a flat note during an update to the game board instead of using an audio codec to encode and decode a sampled sound.

Challenges

Due to some unforeseen circumstances, we were unable to complete this project to the extent we desired. Due to the COVID-19 crisis, our team was split up geographically, with only two of us possessing the DE1-SoC boards. Thus, testing of our hardware was very bottlenecked. Kevin's home router had faulty ethernet connection, meaning it was nearly impossible for him to transfer files to the SD card on the board. Arsalaan's compilation toolchain broke multiple times during this project because of the lack of vm computing and disk resources to run quartus and qsys, but had a working connection to an FPGA board, meaning others had to send him a compiled image to test, severely slowing down the development process.

In addition, we had shipping troubles, since Kevin shipped the joystick controller we were originally going to use to his school address, and it was returned to Amazon since he had already checked out. We spent some time acquiring monitors to use with the FPGA. Due to time constraints at the end and lack of speakers connectable with 3.5mm audio cables, we were unable to test the sound component of our project.

Authorship

Arsalaan wrote the audio code (music.sv) and ioctls on the software side (vga_ball.c, vga_ball.h) to communicate with the hardware. Sooyeon and Kevin worked together to write the game logic in C (tetris.c, tetris.h, hello.c). Kevin wrote the SystemVerilog code for rendering the blocks to the screen (vga_ball.sv). Josh helped with integrating the game logic and driver, as well as setting up the connections between the input/outputs of the modules (qsys).

Final Thoughts and Takeaways

This project was a major challenge for us in many ways. All of us were previously unfamiliar with using HDLs and hardware in general. Just understanding how the modules worked and communicated with each other and with software was a major barrier for most of us. That knowledge and intuition will definitely come in useful in future projects. Since all four of us are going into careers in software engineering where we will come across low level components that may interface with hardware, we anticipate being able to draw on this experience to help our careers in technology. Even if we aren't implementing hardware components, we will be able to understand at a high level how hardware engineers we may work with create their components.

While the end of this project wasn't ideal, we learned a lot about how to approach a hardware project (it is very different from software)! As an advice for future projects, make sure to resolve issues you may run into early on—a broken compilation toolchain or unresponsive terminal will severely hamper your progress. Make sure to allocate adequate time for something that you may think is trivial, even on the software side, but debugging in an environment where it is impossible to log information or set breakpoints will make what appears to be a trivial task much more time consuming.

Files

vga_ball.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic        clk,
             input logic     reset,
          input logic [31:0]  writedata,
          input logic        write,
          input              chipselect,
          input logic [2:0]  address,

          output logic [7:0] VGA_R, VGA_G, VGA_B,
          output logic       VGA_CLK, VGA_HS, VGA_VS,
                             VGA_BLANK_n,
          output logic       VGA_SYNC_n);

   logic [10:0]    hcount;
   logic [9:0]     vcount;
   // logic [7:0]       background_r, background_g, background_b;

   logic [31:0] coords;

   vga_counters counters(.clk50(clk), .*);

   always_ff @(posedge clk)
/*     if (reset) begin
      background_r <= 8'h0;
      background_g <= 8'h0;
      background_b <= 8'h80;
     end else */
     if (chipselect && write)
      begin
           coords[31:0] <= writedata[31:2];
      end

   always_comb begin
      {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
if (vcount >= 50 && vcount < 90 && hcount >= 100 && hcount < 140 &&
coords[0])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 90 && hcount >= 140 && hcount < 180
&& coords[1])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 90 && hcount >= 180 && hcount < 220
&& coords[2])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 90 && hcount >= 220 && hcount < 260
&& coords[3])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 90 && hcount >= 260 && hcount < 300
&& coords[4])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 90 && hcount >= 300 && hcount < 340
&& coords[5])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 100 && hcount <
140 && coords[6])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 140 && hcount <
180 && coords[7])
```

```
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 180 && hcount <
220 && coords[8])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 220 && hcount <
260 && coords[9])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 260 && hcount <
300 && coords[10])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 90 && vcount < 130 && hcount >= 300 && hcount <
340 && coords[11])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 100 && hcount <
140 && coords[12])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 140 && hcount <
180 && coords[13])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 180 && hcount <
220 && coords[14])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 220 && hcount <
260 && coords[15])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 260 && hcount <
300 && coords[16])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 130 && vcount < 170 && hcount >= 300 && hcount <
340 && coords[17])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 170 && vcount < 210 && hcount >= 100 && hcount <
140 && coords[18])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 170 && vcount < 210 && hcount >= 140 && hcount <
180 && coords[19])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 170 && vcount < 210 && hcount >= 180 && hcount <
220 && coords[20])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 170 && vcount < 210 && hcount >= 220 && hcount <
260 && coords[21])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
```

```
else if (vcount >= 170 && vcount < 210 && hcount >= 260 && hcount <
300 && coords[22])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 170 && vcount < 210 && hcount >= 300 && hcount <
340 && coords[23])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 100 && hcount <
140 && coords[24])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 140 && hcount <
180 && coords[25])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 180 && hcount <
220 && coords[26])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 220 && hcount <
260 && coords[27])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 260 && hcount <
300 && coords[28])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 210 && vcount < 250 && hcount >= 300 && hcount <
340 && coords[29])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 100 && hcount <
140 && coords[30])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 140 && hcount <
180 && coords[31])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 180 && hcount <
220 && coords[32])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 220 && hcount <
260 && coords[33])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 260 && hcount <
300 && coords[34])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 250 && vcount < 290 && hcount >= 300 && hcount <
340 && coords[35])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
```

```
else if (vcount >= 290 && vcount < 330 && hcount >= 100 && hcount <
140 && coords[36])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 290 && vcount < 330 && hcount >= 140 && hcount <
180 && coords[37])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 290 && vcount < 330 && hcount >= 180 && hcount <
220 && coords[38])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 290 && vcount < 330 && hcount >= 220 && hcount <
260 && coords[39])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 290 && vcount < 330 && hcount >= 260 && hcount <
300 && coords[40])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 290 && vcount < 330 && hcount >= 300 && hcount <
340 && coords[41])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 100 && hcount <
140 && coords[42])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 140 && hcount <
180 && coords[43])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 180 && hcount <
220 && coords[44])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 220 && hcount <
260 && coords[45])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 260 && hcount <
300 && coords[46])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 330 && vcount < 370 && hcount >= 300 && hcount <
340 && coords[47])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 370 && vcount < 410 && hcount >= 100 && hcount <
140 && coords[48])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 370 && vcount < 410 && hcount >= 140 && hcount <
180 && coords[49])
            {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
```

```
else if (vcount >= 370 && vcount < 410 && hcount >= 180 && hcount <
220 && coords[50])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 370 && vcount < 410 && hcount >= 220 && hcount <
260 && coords[51])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 370 && vcount < 410 && hcount >= 260 && hcount <
300 && coords[52])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 370 && vcount < 410 && hcount >= 300 && hcount <
340 && coords[53])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 100 && hcount <
140 && coords[54])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 140 && hcount <
180 && coords[55])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 180 && hcount <
220 && coords[56])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 220 && hcount <
260 && coords[57])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 260 && hcount <
300 && coords[58])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 410 && vcount < 450 && hcount >= 300 && hcount <
340 && coords[59])
           {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h00, 8'h00};
else if (vcount >= 50 && vcount < 450 && hcount >= 100 && hcount <
340)
           {VGA_R, VGA_G, VGA_B} = {8'h40, 8'h40, 8'h40};
     else
       {VGA_R, VGA_G, VGA_B} =
             {8'h0, 8'h0, 8'h80};
   end

endmodule

module vga_counters(
 input logic         clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other
cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                            VBACK_PORCH; // 525

   logic endOfLine;

   always_ff @(posedge clk50 or posedge reset)
     if (reset)          hcount <= 0;
     else if (endOfLine) hcount <= 0;
     else                hcount <= hcount + 11'd 1;

   assign endOfLine = hcount == HTOTAL - 1;

   logic endOfField;
   always_ff @(posedge clk50 or posedge reset)
     if (reset)          vcount <= 0;
     else if (endOfLine)
       if (endOfField)   vcount <= 0;
       else              vcount <= vcount + 10'd 1;

   assign endOfField = vcount == VTOTAL - 1;

   // Horizontal sync: from 0x520 to 0x5DF (0x57F)
   // 101 0010 0000 to 101 1101 1111
   assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                 !(hcount[7:5] == 3'b111));
   assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

   assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal;
unused

   // Horizontal active: 0 to 1279     Vertical active: 0 to 479
   // 101 0000 0000  1280         01 1110 0000  480
   // 110 0011 1111  1599         10 0000 1100  524
   assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                !( vcount[9] | (vcount[8:5] == 4'b1111) );

   /* VGA_CLK is 25 MHz
    *             __    __    __
    * clk50    __|  |__|  |__|
    *
    *             _____       __
    * hcount[0]__|     |_____|
    */
   assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

music.sv

```
module music(clk50, spk);
input clk50;
output spk;
```

```
parameter clkdivider = 25000000/440/2;

reg [14:0] c;
always @(posedge clk) if(c==0) c <= clkdivider-1; else c <= c-1;

reg speaker;
always @(posedge clk) if(c==0) spk <= ~spk;
endmodule
```

Makefile

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
        obj-m := vga_ball.o

else

HELLO_OBJ = hello.o tetris.o usbkeyboard.o

# We are being compiled as a module: use the Kernel build system

     KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
        PWD := $(shell pwd)

default: module hello

module:
     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

hello: $(HELLO_OBJ)
     cc -Wall -o hello $(HELLO_OBJ) -lusb-1.0 -pthread
hello.o : hello.c tetris.h usbkeyboard.h
tetris.o : tetris.c tetris.h
usbkeyboard.o : usbkeyboard.c usbkeyboard.h

clean:
     ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
     ${RM} hello

TARFILES = Makefile README vga_ball.h vga_ball.c hello.c
```

```
TARFILE = lab3-sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
	tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)

endif
```

vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>
#include <asm-generic/int-ll64.h>


typedef struct {
	__u32 bits;
} vga_ball_bitmap_t;


typedef struct {
  vga_ball_bitmap_t bitmap;
} vga_ball_arg_t;


#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1,
vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2,
vga_ball_arg_t *)

#endif
```

vga_ball.c

```
/* * Device driver for the VGA video generator
```

```
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/delay.h>
#include <asm-generic/int-ll64.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
/**#define BG_RED(x) (x)
#define BG_GREEN(x) ((x)+1)
#define BG_BLUE(x) ((x)+2)
**/
```

```
#define BIT_STORE(x) (x)

/*
 * Information about our device
 */
struct vga_ball_dev {
     struct resource res; /* Resource: our registers */
     void __iomem *virtbase; /* Where registers can be accessed in
memory */
        vga_ball_bitmap_t bitmap;
           // vga_ball_bitmap2_t bitmap2;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set
up
 */
static void write_background(vga_ball_bitmap_t *bitmap)
{
     // writel(bitmap->bits, BIT_STORE(dev.virtbase));
     iowrite32(bitmap->bits, BIT_STORE(dev.virtbase));
     dev.bitmap = *bitmap;
}

/** static void write_background2ivga_ball_bitmap2_t *bitmap2){
     writel(bitmap->bits, BIT_STORE(dev.virtbase));
     dev.bitmap = *bitmap;
} **/



/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned
long arg)
{
     vga_ball_arg_t vla;

     switch (cmd) {
```

```
	case VGA_BALL_WRITE_BACKGROUND:
		if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
				   sizeof(vga_ball_arg_t)))
			return -EACCES;
		write_background(&vla.bitmap);
		break;

	case VGA_BALL_READ_BACKGROUND:
		vla.bitmap = dev.bitmap;
		if (copy_to_user((vga_ball_arg_t *) arg, &vla,
				 sizeof(vga_ball_arg_t)))
			return -EACCES;
		break;
	/** case VGA_BALL_WRITE_BACKGROUND2:
		if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
				   sizeof(vga_ball_arg_t)))
			return -EACCES;
		write_background2(&vla.bitmap2);
		break;
		**/

	default:
		return -EINVAL;
	}

	return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a
char dev */
static struct miscdevice vga_ball_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &vga_ball_fops,
};

/*
```

```
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{



     // vga_ball_bitmap_t bittest = { 0U };

     int ret;

     /* Register ourselves as a misc device: creates /dev/vga_ball
*/
     ret = misc_register(&vga_ball_misc_device);

     /* Get the address of our registers from the device tree */
     ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
     if (ret) {
          ret = -ENOENT;
          goto out_deregister;
     }

     /* Make sure we can use these registers */
     if (request_mem_region(dev.res.start, resource_size(&dev.res),
                    DRIVER_NAME) == NULL) {
          ret = -EBUSY;
          goto out_deregister;
     }

     /* Arrange access to our registers */
     dev.virtbase = of_iomap(pdev->dev.of_node, 0);
     if (dev.virtbase == NULL) {
          ret = -ENOMEM;
          goto out_release_mem_region;
     }

     /* Set an initial color */
    // write_background(&bittest);

     return 0;

out_release_mem_region:
	release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
	misc_deregister(&vga_ball_misc_device);
	return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
	iounmap(dev.virtbase);
	release_mem_region(dev.res.start, resource_size(&dev.res));
	misc_deregister(&vga_ball_misc_device);
	return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
	{ .compatible = "csee4840,vga_ball-1.0" },
	{},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
	.driver	= {
		.name	= DRIVER_NAME,
		.owner	= THIS_MODULE,
		.of_match_table = of_match_ptr(vga_ball_of_match),
	},
	.remove	= __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
	pr_info(DRIVER_NAME ": init\n");
	return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
```

```
{
     platform_driver_unregister(&vga_ball_driver);
     pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

hello.c

```
/*
 * Userspace program that communicates with the vga_ball device
driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include <stdlib.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include <pthread.h>
#include "tetris.h"

#include <asm-generic/int-ll64.h>
#include "usbkeyboard.h"


int vga_ball_fd;
```

```
struct libusb_device_handle *keyboard;
uint8_t endpoint_address;


/* Read and print the background color */
void print_background_color() {
  vga_ball_arg_t vla;

  if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
      return;
  }
  printf("%lu\n",
       vla.bitmap.bits);
}

/* Set the background color */
void set_background_color(vga_ball_bitmap_t *c)
{
  vga_ball_arg_t vla;
  vla.bitmap = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
      return;
  }
}

void write_state(board_t state)
{
  vga_ball_bitmap_t test = {state};
  set_background_color(&test);
  print_background_color();
}

struct thread_info {
  int movement;
};

void *
keyboard_thread_f (void *args_v) {
  struct thread_info *args = (struct thread_info *)args_v;
  struct usb_keyboard_packet packet;
  int transferred;
  char keystate[12];

    /* Open the keyboard */
  if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
    fprintf(stderr, "Did not find a keyboard\n");
    exit(1);
  }

  for(;;) {

    libusb_interrupt_transfer(keyboard, endpoint_address,
      (unsigned char *) &packet, sizeof(packet),
      &transferred, 0);
    if (transferred == sizeof(packet)) {
      sprintf(keystate, "%02x %02x %02x", packet.modifiers,
packet.keycode[0],
        packet.keycode[1]);
      if (packet.keycode[0] == 80) { // Left
        printf("LEFT!!!!!\n");
        args->movement = 1;
      } else if (packet.keycode[0] == 79) { // Right
        printf("RIGHTT!!!!\n");
        args->movement = 2;

        }
    }
  }
}

void play_tetris()
{

  pthread_t keyboard_thread;

  struct thread_info *args = malloc(sizeof(struct thread_info));
  args->movement = 0;

  pthread_create(&keyboard_thread, 0, &keyboard_thread_f, args);

  int board[HEIGHT][WIDTH];
  int piece[HEIGHT][WIDTH];
  board_t res;
  memset (&board, 0, sizeof (int) * HEIGHT * WIDTH);
  res = piece_and_board (&board, &piece);
  write_state (res);
  new_piece (&piece);
  for (int i = 0;;i += 500)
    {
      if (args->movement) {
        if (args->movement == 1) {
          piece_left(&board, &piece);
        }
        else if (args->movement == 2) {
          piece_right(&board, &piece);
        }
        args->movement = 0;
        res = piece_and_board (&board, &piece);
        write_state (res);
        print_piece_and_board(res);
      }

      if (i % 1000000 == 0) {
        if (collision (&board, &piece))
        {
          settle_piece (&board, &piece);
          new_piece (&piece);
        }
            else
        {
          piece_down (&piece);
        }

        res = piece_and_board (&board, &piece);
        write_state (res);
        print_piece_and_board(res);
      }

      usleep (500);
    }
}

int main()
{
  vga_ball_arg_t vla;
  int i;
  static const char filename[] = "/dev/vga_ball";


  /** if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
    fprintf(stderr, "Did not find a keyboard\n");
    exit(1);
  } **/


  // static const vga_ball_color_t colors[] = {
  //   { 0xff, 0x00, 0x00 }, /* Red */
  //   { 0x00, 0xff, 0x00 }, /* Green */
  //   { 0x00, 0x00, 0xff }, /* Blue */
  //   { 0xff, 0xff, 0x00 }, /* Yellow */
  //   { 0x00, 0xff, 0xff }, /* Cyan */
  //   { 0xff, 0x00, 0xff }, /* Magenta */
  //   { 0x80, 0x80, 0x80 }, /* Gray */
  //   { 0x00, 0x00, 0x00 }, /* Black */
  //   { 0xff, 0xff, 0xff }  /* White */
  // };

//vga_ball_bitmap_t test = {430};

# define COLORS 9

  printf("VGA ball Userspace program started\n");

  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }

  printf("initial state: ");

  /** for(;;){
    libusb_interrupt_transfer(keyboard, endpoint_address,
          (unsigned char *) &packet, sizeof(packet),
          &transferred, 0);
    if (transferred == sizeof(packet)){
      offset = 93;

      keycode0Int = packet.keycode[0] + offset;
      keycode1Int = packet.keycode[1] + offset;
      if (keycode0Int == 133){ // Enter pressed
          printf("Enter pressed\n");
          //set_background_color(&test[0]);
      }
    }

  } **/

  play_tetris();


  printf("VGA BALL Userspace program terminating\n");
  return 0;
}
```

tetris.h

```
#include <stdio.h>
#include <stdlib.h>
#include <asm-generic/int-ll64.h>
#include <unistd.h>
#include <time.h>

#define WIDTH 6
#define HEIGHT 5

typedef __u32 board_t;

board_t
piece_and_board (int (*piece)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH]);

void
print_matrix (int (*matrix)[HEIGHT][WIDTH]);

void
print_piece_and_board (board_t res);

void
```

```
settle_piece (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH]);

int
collision (int (*board)[HEIGHT][WIDTH], int (*piece)[HEIGHT][WIDTH]);

void
piece_down (int (*piece)[HEIGHT][WIDTH]);

void
piece_left (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH]);

void
piece_right (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH]);

void
new_piece (int (*piece)[HEIGHT][WIDTH]);
```

tetris.c

```
// Author: Kevin Li

#include <stdio.h>
#include <stdlib.h>
#include <asm-generic/int-ll64.h>
#include <unistd.h>
#include <time.h>
#include "tetris.h"
#include <string.h>

board_t
piece_and_board (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH])
{
  board_t res = 0;
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
       res |= (((*board)[i][j] | (*piece)[i][j]) & 1UL) << (i *
WIDTH + j);
     }
    }
  return res;
}

void
print_matrix (int (*matrix)[HEIGHT][WIDTH])
{
  printf ("*******\n");
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
     {
       printf ("%d", (*matrix)[i][j]);
     }
      printf ("\n");
    }
  printf ("*******\n");
}

void
print_piece_and_board (board_t res)
{
  printf ("%lu\n", res);
  printf ("--------\n");
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
     {
       if ((res >> (i * WIDTH + j)) & 1UL)
         printf ("1");
       else
         printf ("0");
     }
      printf ("\n");
    }
  printf ("--------\n");
}

void
```

```
settle_piece (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH])
{
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if ((*piece)[i][j])
          (*board)[i][j] |= 1;
      }
    }
}

int
collision (int (*board)[HEIGHT][WIDTH], int (*piece)[HEIGHT][WIDTH])
{
  for (int i = 0; i < HEIGHT - 1; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if ((*board)[i + 1][j] && (*piece)[i][j])
          {
            return 1;
          }
      }
    }
  for (int j = 0; j < WIDTH; ++j)
    {
      if ((*piece)[HEIGHT - 1][j])
      {
        return 1;
      }
    }
  return 0;
}

void
piece_down (int (*piece)[HEIGHT][WIDTH])
{
  int nextpiece[HEIGHT][WIDTH];
  memset (nextpiece, 0, sizeof (int) * HEIGHT * WIDTH);
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if ((*piece)[i][j])
          {
            nextpiece[i + 1][j] = 1;
          }
      }
    }
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if (nextpiece[i][j])
          {
            (*piece)[i][j] = 1;
          }
        else
          {
            (*piece)[i][j] = 0;
          }
      }
    }
}

void
piece_left (int (*board)[HEIGHT][WIDTH], int (*piece)[HEIGHT][WIDTH])
{
  int nextpiece[HEIGHT][WIDTH];
  memset (nextpiece, 0, sizeof (int) * HEIGHT * WIDTH);
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if ((*piece)[i][j])
          {
            if (j == 0 || (*board)[i][j-1])
           return;
            nextpiece[i][j - 1] = 1;
          }
      }
    }
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if (nextpiece[i][j])
          {
            (*piece)[i][j] = 1;
          }
        else
          {
            (*piece)[i][j] = 0;
          }
      }
    }
}

void
piece_right (int (*board)[HEIGHT][WIDTH], int
(*piece)[HEIGHT][WIDTH])
{
  int nextpiece[HEIGHT][WIDTH];
  memset (nextpiece, 0, sizeof (int) * HEIGHT * WIDTH);
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if ((*piece)[i][j])
          {
            if (j == WIDTH - 1 || (*board)[i][j+1])
           return;
            nextpiece[i][j + 1] = 1;
          }
      }
    }
  for (int i = 0; i < HEIGHT; ++i)
    {
      for (int j = 0; j < WIDTH; ++j)
      {
        if (nextpiece[i][j])
          {
            (*piece)[i][j] = 1;
          }
        else
          {
            (*piece)[i][j] = 0;
            }
        }
      }
}

void
new_piece (int (*piece)[HEIGHT][WIDTH])
{
  int random;
  random = rand () % 7;
  printf ("rand: %d\n", random);
  memset (*piece, 0, sizeof (int) * HEIGHT * WIDTH);
  switch (random)
    {
    case 0:                 // O
      (*piece)[0][0] = (*piece)[0][1] = (*piece)[1][0] =
(*piece)[1][1] = 1;
      break;
    case 1:                 // T
      (*piece)[0][1] = (*piece)[1][0] = (*piece)[1][1] =
(*piece)[1][2] = 1;
      break;
    case 2:                 // L
      (*piece)[0][0] = (*piece)[1][0] = (*piece)[1][1] =
(*piece)[1][2] = 1;
      break;
    case 3:                 // J
      (*piece)[0][2] = (*piece)[1][0] = (*piece)[1][1] =
(*piece)[1][2] = 1;
      break;
    case 4:                 // I
      (*piece)[0][0] = (*piece)[0][1] = (*piece)[0][2] =
(*piece)[0][3] = 1;
      break;
    case 5:                 // S
      (*piece)[0][0] = (*piece)[0][1] = (*piece)[1][1] =
(*piece)[1][2] = 1;
      break;
    case 6:                 // Z
      (*piece)[0][0] = (*piece)[0][1] = (*piece)[1][1] =
(*piece)[1][2] = 1;
      break;
    }
```

```
}
```