Jeff Heckey
ECE 253
12/12/13

Spartan Tetris

Sources

https://github.com/jheckey/spartan_tetris

Concept

Implement Tetris on a Spartan 1600E Starter Kit. This involves developing a new VGA Pcore for integrating into a MicroBlaze system, and writing software to control the Pcore and implement the game mechanics.

Design

- VGA output
- Clickable rotary-encoder controller
- 20x10 game board
- Display next piece, score, level
- Random piece generation
- Level increases, reduces fall time
- Pieces fall, lines will be detected and cleared

Plan

1. VGA display output (**Wed 11/13**)
    1. Get VGA pcore instantiated and able to output to a screen
    2. Instantiate all support modules (RAMs, etc.)
2. Control display (**Sat 11/16**)
    1. Change colors automatically from the program
    2. -OR- display text
    3. Change periodically

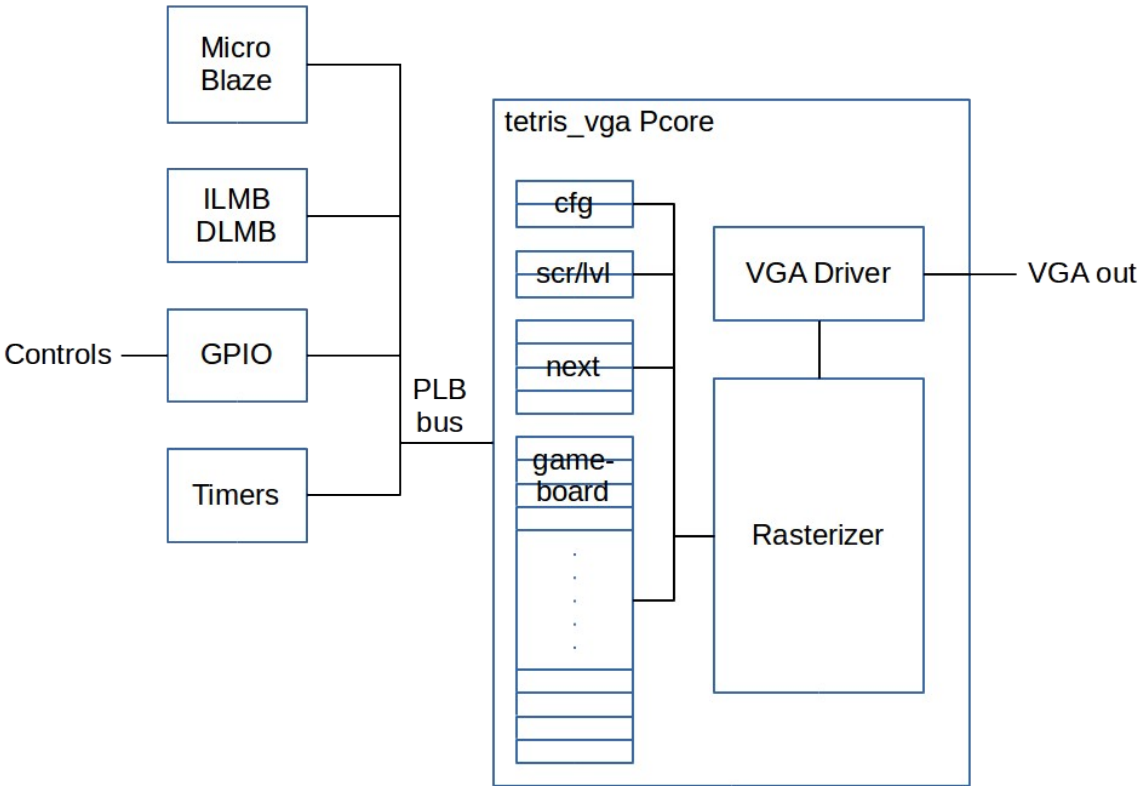
3. Get inputs (**Sun 11/17**)
    1. React to interrupts from rotary encoder by changing LEDs
4. Use inputs to control display (**Tue 11/19**)
    1. Change the colors/text on the display on input
5. Draw a static scene (**Thu 11/21**)
    1. Demonstrate non-uniform output from VGA controller
    2. Multiple values in VRAM
6. Draw a moving object (**Fri 11/22**)
    1. Use MicroBlaze to dynamically draw a moving object (pong ball)
    2. Demonstrate clean refresh
7. Control a moving object with inputs (**Sun 11/24**)
    1. Use interrupts to move an object on the screen
    2. Demonstrate control of VRAM
    3. Demonstrate clean refresh
8. Draw game board and pieces (**Tue 11/26**)
    1. Implement data structures and basic units for Tetris
9. Make tetramino fall (**Fri 11/29**)
    1. Integrate movement with game board and non-uniform shapes
    2. Stop block at the right elevation
    3. Show control over display and object motion
10. Make tetramino move (**Sat 11/30**)
    1. Integrate the controls into main loop
    2. Update the movement without tearing
    3. Stop the tetramino at the right elevation
11. Make tetramino rotate (**Sun 12/1**)
    1. Flip it without poor updating
12. Color pieces (**Tue 12/3**)
    1. Oooh! Pretty!
13. Clear lines (**Thu 12/5**)
    1. Detect complete line(s)
    2. Large screen update (drop floating blocks)
14. Add scoring (**Sat 12/7**)
    1. Calculate and track score
    2. Display score to screen

Implementation

The implementation consists of two distinct components, hardware and software. The hardware components consisted of a standard MicroBlaze System Builder layout, but incorporated a new pcore to drive the VGA. The software was built mostly from scratch, but uses the standard libraries and reuses the encoder debouncing from lab 2.

Hardware

The hardware layout is shown below.



The VGA pcore is broken into 3 components, a bus interface containing the registers, a rasterizer, and a VGA driver. The VGA driver draws a single pixel as a three bit RGB value, creating 8 colors. The clock 50MHz. 16 pixels (48 bits) are read in each drawn pixel. The x and y coordinates are tracked with individual counters; the x counter increments each clock and is downsampled to the pixel clock of 25MHz by ignoring the LSB, the y counter increments whenever the x counter wraps at the end of the line (x=1600). The y counter wraps at 521.

The rasterizer draws 16 pixels at a time for the VGA driver base on a 40x30 tile layout of the screen. Each tile is 16x16 pixels. The tiles are used to orient the rasterizer and allow for drawing the next tetramino (game piece) and gameboard from the CPU accessible registers in the pcore. Boarders are hardcoded into the rasterizer. The logic here was extremely difficult to layout properly in order to allow for decent pipelining. I had to break the steps into several pipeline stages in order for the logic to be placed in the FPGA. This resulted in a 4 stage pipeline where the first stage determines the current tile value and the tile row look at, the second determines which tile column to use, the third converts that

tile glyph to a color, and the fourth renders the pixels. The glyph update wound up being unused due to time constraints. The pixel logic was fairly complicated in order to encode the boarders, but the tetramino tiles were fairly straight forward.

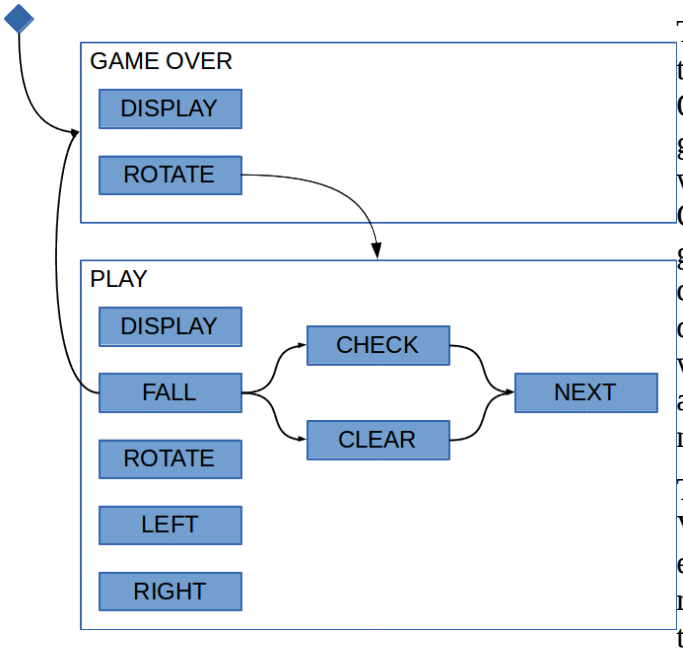
The pcore wrapper was generated with 58 registers. The first 2 are configuration and debug registers for interacting with the VGA driver. The next 50 registers are divided into 200 glyph bytes for the gameboard. The next 4 registers were 16 glyphs for the next tetramino display. The last 2 were for the score and level, but were never implemented due to time constraints.

The most challenging part of building the pcore was actually just getting code changes to work. For a while I had to do my development in ISE and download with Impact to test early changes. After that, I tried creating a new version of the pcore to update in XPS. Eventually I found that the most reliable way to pull in updates was to do a “Hardware->Clean Hardware” then “Export Hardware Design to SDK”.

With some more time, I would have included hardware debouncing to improve the controls, which can be glitchy currently.

Software

The software was initially going to be implemented with QP-Nano, but the memory footprint was too large. Instead I went with a priority event queue. Each interrupt handler would set an action in the queue and the main code would poll for new actions. Once a new action was seen, that one action would be executed, though it can set another action if needed. Se the HFSM below for how actions are handled.



The FALL action will typically just drop the tetramino one tile, but it will end the game (GAME OVER) if the tile is stopped at the top of the gameboard. If the tetramino has hit the bottom, it will add CHECK_LINES to the action queue. CHECK_LINES will check for any lines in the gameboard and clear them from the gameboard display if needed, otherwise the next piece will be dropped. If there were lines cleared, the next FALL will cause the will go to clear the lines, and update all the tiles in the gameboard, then will drop the next piece.

The interrupt handlers will respond to the VGA VSYNC signal, the fall timer, and click and rotate events from the GPIO. The VGA VSYNC forces a redraw of the gameboard from the in-memory state to the screen; this take about 0.1 ms, which is small enough to do during the time to go from the lower right to the upper left corner. The fall timer handler will update the fall timer counter value, so that stays in sync. The click and rotate events will trigger timer-based debounced code. Once the event is debounced, the appropriate click (ROTATE) or LEFT or RIGHT action is added to the action queue.

There were some severe difficulties with figuring out the VGA pcore interrupt controls. Mainly, I found that it is very carefully controlled and I expected it to be harder, winding up digging through the HDL and C code. Part of the problem was that after updating the version of the hardware, the BSP files

hadn't been regenerated, so the interrupt register offsets were incorrect. That meant that the supplied driver code was incorrect. After regenerating it worked perfectly.