# FPGA Tetris Game Implementation

# 1) Introduction:

Tetris is a classic puzzle game that has been a staple in gaming for decades. In this project, we've implemented a hardware-accelerated version of Tetris using an Urbana FPGA board with System-on-Chip (SoC) capabilities. The core objective was to create a responsive and visually appealing Tetris game that leverages the parallel processing capabilities of the Urbana FPGA while maintaining the familiar gameplay mechanics that have made Tetris so enduring.

**Project Summary – Two-Player SoC Tetris FPGA (Technical Pitch)**

Our FPGA Tetris system leverages a Xilinx Spartan-7 to deliver a full System-on-Chip gaming experience complete with real-time graphics, control, and audio. The design features VGA-based hardware-accelerated rendering of the game board, driven by a custom video pipeline in programmable logic for smooth tile graphics. A MicroBlaze soft-core processor is integrated as the brains of the system, handling high-level game logic and orchestration alongside the FPGA's hardware accelerators. The graphics subsystem employs a dual-ROM architecture (for color palette and tetromino shape data) to efficiently generate video frames, which are output in VGA timing and then converted via a VGA-to-HDMI interface for display on modern screens. Interactive gameplay is enabled through a USB keyboard interface using an AXI UARTlite core, providing low-latency capture of player inputs over a standard USB-UART bridge. On the audio side, the project includes an AXI-driven sound module: an AXI SPI IP core communicates with an on-board audio codec or waveform generator while a PWM IP module outputs analog audio, together producing classic Tetris music and sound effects. All these components are seamlessly connected via the AXI interconnect, showcasing a sophisticated embedded architecture within the FPGA.

Advanced digital design techniques ensure the game runs reliably and in sync with external displays, while also enabling an innovative multiplayer feature. A dedicated clock division logic ties the game's timing to the video vsync signal, establishing a stable frame rate and a deterministic "tick" for piece movement and gravity. This guarantees that rendering and gameplay updates remain in lockstep (preventing tearing and ensuring consistent piece fall rates). The Tetris board state is stored in BRAM (on-chip Block RAM), allowing rapid concurrent access by both the MicroBlaze and the rendering hardware – a true dual-port memory design that eliminates bottlenecks when reading or modifying the playfield matrix. The result is flicker-free graphics and responsive gameplay logic all contained within FPGA fabric. Crucially, the project extends beyond single-player: it incorporates multiplayer support over Ethernet via UDP/IP using lwIP on MicroBlaze. Two FPGA boards can be networked, with the MicroBlaze processors exchanging game state messages through a low-latency UDP protocol. The

lightweight IP stack (lwIP) running on each MicroBlaze manages the networking behind the scenes, so the two-player Tetris mode remains fast and synchronized. In summary, this project demonstrates a cutting-edge FPGA-based game console: combining hardware-accelerated VGA rendering, a soft-core CPU for control, AXI-connected peripherals (video, audio, input), precise timing control, and real-time multiplayer networking, all in one impressive package.

**Extending a Spartan-7 FPGA Tetris Game to Multiplayer Over Ethernet**

**Ethernet Interface Options for FPGA Multiplayer**

To enable two Spartan-7 FPGA boards to communicate for multiplayer Tetris, the primary requirement is an Ethernet interface accessible by the MicroBlaze soft processor. Two practical options exist: using a Pmod-based Ethernet controller module or leveraging Xilinx's AXI Ethernet Lite IP core. The choice affects both hardware integration and software stack development.

**Pmod Ethernet Controller (e.g., W5500 or ENC424J600-based NIC – Digilent Pmod NIC100)**

This approach uses an external Pmod module that provides a complete 10/100 Mbps Ethernet controller with integrated MAC and PHY. Modules like the NIC100 use an SPI interface (via an AXI Quad SPI IP in the FPGA design) to communicate with the MicroBlaze, simplifying hardware connections. For example, the WIZnet W5500 chip includes a built-in TCP/IP stack and a fully integrated 10/100 Ethernet PHY, meaning much of the protocol handling is offloaded to hardware. Only high-level socket commands (through SPI) are needed to send/receive data. The FPGA logic cost is minimal (just the SPI controller and a few GPIO pins for interrupts/resets), and it can easily be added to a Vivado Block Design. A trade-off arises when the chip (like the older Microchip ENC424J600 on NIC100) lacks a full TCP/IP stack, in which case the MicroBlaze must run a network stack in software. Regardless, the Pmod still provides out-of-the-box MAC/PHY functionality. Overall, a Pmod NIC is practical for boards without built-in Ethernet, requiring only a Pmod header, SPI IP core, and interrupt line, with no need for MII/RMII connections or PCB magnetics.

**AXI Ethernet Lite MAC IP Core**

Xilinx offers a lightweight Ethernet MAC (10/100 Mbps) instantiated in the FPGA fabric and controlled by MicroBlaze over AXI4. The AXI Ethernet Lite core, designed for minimal resource usage, is straightforward to set up in Vivado's IP Integrator. It connects to the processor via memory-mapped AXI and presents simple transmit/receive buffer interfaces. One advantage is its integration with Xilinx's lwIP software stack, enabling the MicroBlaze to use standard Ethernet APIs or sockets. However, this core requires an external PHY chip, connected via MII signals to the FPGA. This means routing MII (or RMII) pins to an Ethernet PHY transceiver,

often via board headers or an add-on module, along with an MDIO interface for PHY management. For example, using AXI Ethernet Lite without a built-in Ethernet port may involve attaching a 10/100 Mbps PHY (like DP83848 or KSZ8081) on a custom adapter. A 25 MHz reference clock must also be supplied to the PHY/MAC. This additional hardware complexity makes setup less plug-and-play than SPI modules. The AXI Ethernet Lite core operates at a 100 MHz AXI clock and includes small internal dual-port RAMs for buffering Ethernet frames. It provides interrupt lines for transmit/receive events. The typical workflow involves enabling the lwIP library in the MicroBlaze BSP, which supplies drivers (xemaclite) and TCP/IP support.

**Recommendation**

For a Spartan-7 board without Ethernet, a W5500-based Pmod NIC is the most practical Ethernet solution. It simplifies the FPGA design and manages MAC/PHY duties in hardware, reducing time spent on constraints and PHY setup. Developers can focus on writing C code to exchange game data. Conversely, if the setup includes a usable Ethernet PHY, the AXI Ethernet Lite core integrates cleanly with MicroBlaze and lwIP, easing software development. Both options support 10/100 Mbps Ethernet, sufficient for Tetris data needs.

**Synchronizing Game State Between Two FPGAs**

To enable multiplayer Tetris on two FPGA boards, game state or player actions must be communicated over the network with minimal delay. The goal is synchronized game instances or versus mode interactions. A UDP-based protocol is ideal due to its lower latency and overhead compared to TCP. UDP sends packets without delivery guarantees, avoiding retransmission delays and in-order delivery enforcement that could stall gameplay. It fits Tetris well, where frequent updates (piece movements, scores) can correct missed packets.

UDP/IP can run on MicroBlaze using the lwIP stack or the W5500's internal stack. In MicroBlaze + AXI Ethernet setups, lwIP provides UDP APIs via socket or raw interfaces. For W5500, it handles UDP sockets internally and communicates over SPI using socket commands. In both cases, UDP allows simple message passing.

**Data Exchange and Game Logic**

The simplest synchronization method is input exchange. When Player 1 moves or rotates a piece, the MicroBlaze sends a small UDP packet to the other board describing the action. Each FPGA simulates both players' boards, staying in sync if they share the initial state and random piece sequence. This lockstep simulation may include periodic full board snapshots to correct drift.

Alternatively, a state-exchange model has each board simulate its local game and send high-level results (like line clears). For example, clearing lines might trigger a "garbage" message to the

opponent. Boards may also send score or piece info for display in a side panel. This mode is easier to implement but may not maintain identical gameplay states.

A hybrid approach is also viable: share initial seeds and send only player inputs, or run independent games with attack messages. Regardless, minimizing latency is critical. A direct Ethernet link or switch connection yields sub-millisecond latency. The Tetris game should synchronize networking with its frame update loop (e.g., every 16ms for 60 FPS). A typical MicroBlaze loop might read inputs, apply game logic, send network messages via non-blocking UDP, and poll for incoming messages.

Interrupts from Ethernet cores or the W5500 can signal new data, but polling per frame may suffice. Avoid blocking network operations by using non-blocking sockets or lwIP raw API. Each packet can be processed in microseconds; a 20-byte UDP packet transmits in ~1.6 microseconds over 100 Mbps. The system can exchange game data in real time.

**Hardware and IP Blocks Required**

In addition to VGA and audio subsystems, multiplayer support needs:

- Ethernet IP: Either AXI Quad SPI IP + GPIO for W5500/ENC424J600 or AXI Ethernet Lite with an external PHY and MDIO interface. A 25 MHz clock is required for the PHY.

- MicroBlaze firmware updates: Initialize networking, set static IPs, configure lwIP or W5500 via SPI. lwIP use involves calling `lwip_init()` and configuring the netif. For W5500, firmware sets mode registers, IP, subnet, and opens UDP sockets via SPI.

- Communication protocol logic: For lwIP, create a UDP PCB, bind to a port, use `udp_sendto`, and set receive callbacks. For W5500, write to TX buffer, issue SEND, and poll/interrupt on RX. Packet formats can be custom structs or raw bytes (e.g., one-byte event code + data).

- Low-latency game loop design: Integrate networking with game ticks. If using interrupts, set flags and process in the main loop to avoid interrupt overload. Aim for <16 ms latency from action to display. Send crucial messages (like "game over") multiple times or implement acknowledgment.

- Testing and calibration: Begin with test patterns (like LED toggles). Measure round-trip times and jitter. Use Wireshark on a PC for UDP traffic debugging.

This protocol, tightly coupled with the game cycle, allows two FPGAs to remain in sync with responsive multiplayer Tetris. Whether offloaded via the W5500 or handled via AXI Ethernet

Lite, the combination of hardware-based networking and lightweight UDP/IP stack enables real-time, low-latency communication.

# 2) Written Description of Final Project System:

Our Tetris implementation builds upon the foundations established in Lab 6.2, significantly modifying and extending its functionality. We started with the existing codebase from Lab 6.2, which provided basic VGA display capabilities and a simple moving ball example. We transformed this system into a fully functional Tetris game by implementing core gameplay mechanics and visual elements.

The primary modifications to the Lab 6.2 code involved repurposing the ball.sv module to handle Tetris pieces instead of a bouncing ball. We implemented a grid-based system where each cell could be empty or contain a colored block. The Tetris pieces, each composed of four blocks in specific arrangements, move and rotate within this grid according to player input and game rules.

Our Tetris implementation consists of several key components that work together to create a complete gaming system. At the heart of our design is the tetris_core module within tetris.sv, which handles the core game logic such as piece movement, rotation, and collision detection. The tetris_render module, also in tetris.sv, takes care of rendering the game board and active pieces to the display. These hardware modules interact with the MicroBlaze processor through memory-mapped registers, allowing the software to control the game state and receive updates on the game status.

The data flow begins with the USB keyboard input, which is processed by the MicroBlaze processor. The processor translates key presses into game commands (move left/right, rotate, soft drop) and writes these to the memory-mapped registers accessible by the hardware modules. The tetris_core module reads these commands and updates the game state accordingly, handling collision detection and piece locking. The game state is then rendered by the tetris_render module, which outputs the appropriate RGB values to the VGA controller for display on the screen. The game loop operates at 60 Hz, synchronized with the VGA vertical refresh rate, ensuring smooth gameplay.

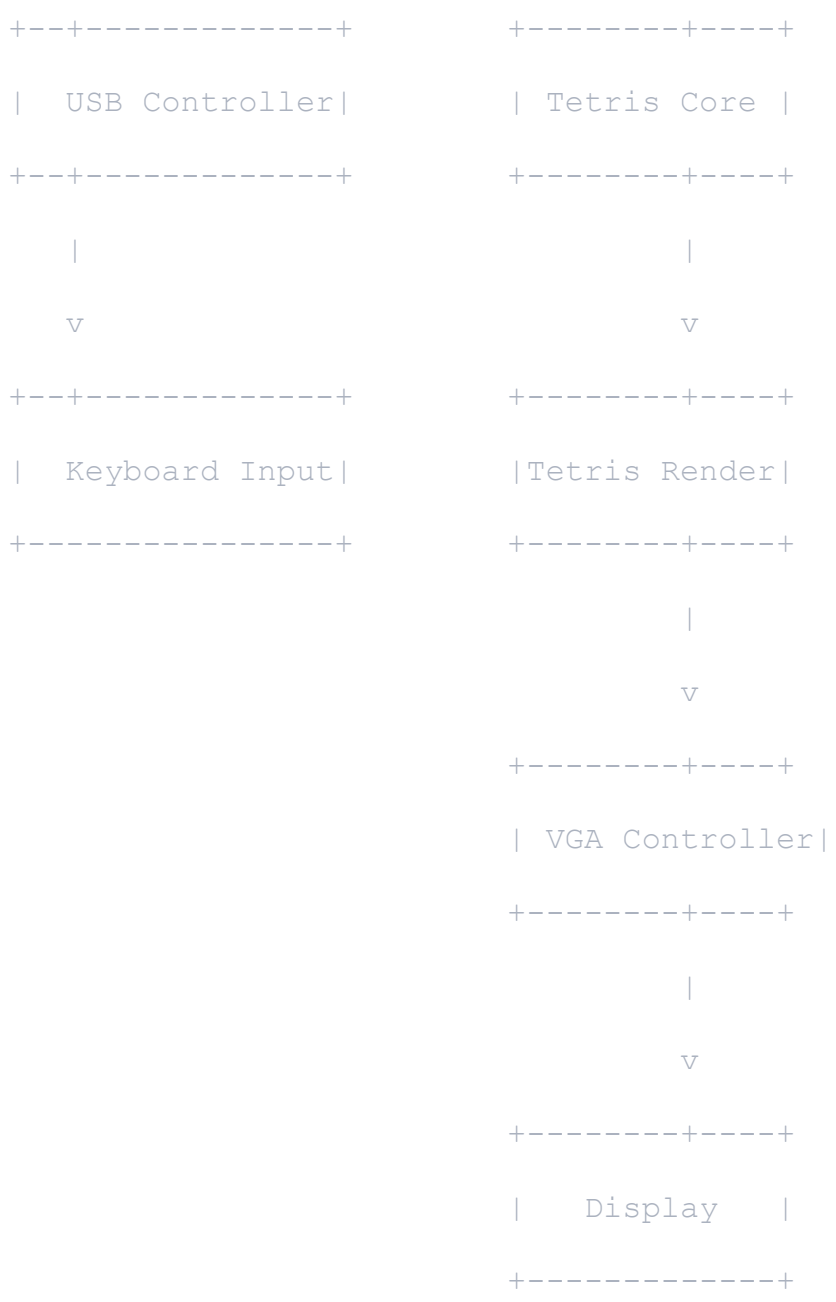
Key features of our implementation include:

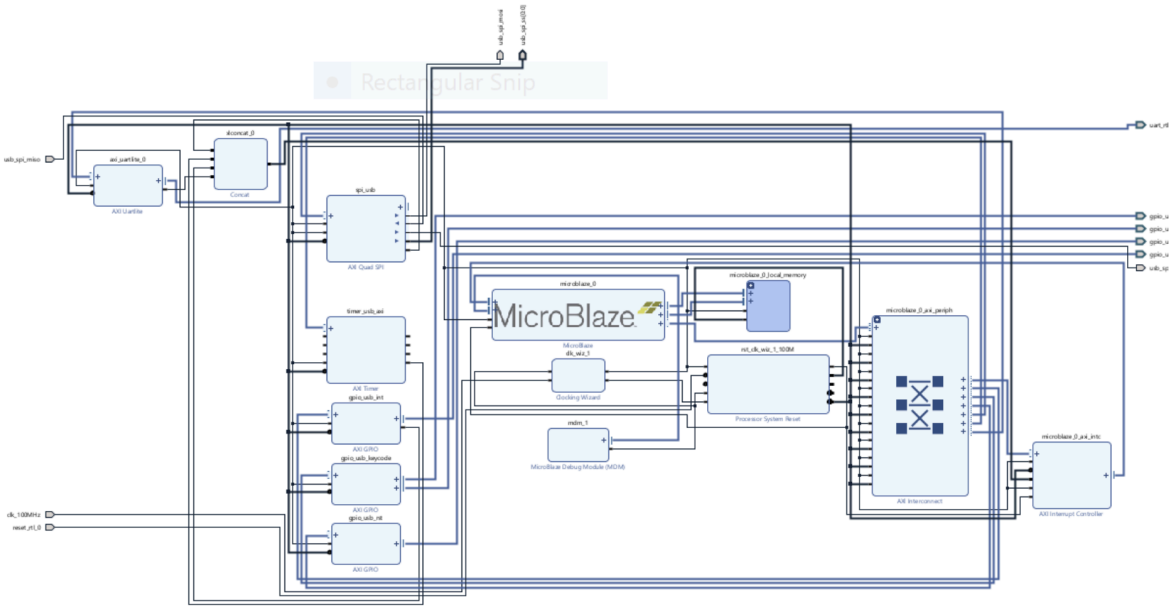
1. **Score Keeping**: We implemented a scoring system that awards points for each cleared line. The score is displayed directly on the Urbana FPGA board using the hexadecimal displays, allowing players to track their progress throughout the game.
2. **Line Clearing**: When a player completes a horizontal line by filling all cells, that line is cleared, and all pieces above it shift down. This classic Tetris mechanic is fully implemented in our system, with appropriate scoring rewards for each cleared line.

3. **Game Over Detection**: Our system detects when pieces stack to the top of the playing field, triggering a game over state. When this occurs, the entire screen turns red, providing immediate visual feedback to the player that the game has concluded.
4. **Piece Movement and Rotation**: Players can move pieces left and right, rotate them, and accelerate their descent using keyboard controls. The system implements proper collision detection to prevent pieces from overlapping or moving outside the boundaries of the game area.

The core gameplay loop involves spawning new Tetris pieces at the top of the screen, allowing the player to control their descent and placement, checking for completed lines, updating the score, and continuing until the game over condition is met. This creates an engaging and responsive gaming experience that faithfully recreates the classic Tetris gameplay.

# 3) Block Diagram


USB Controller
Keyboard Input
Tetris Core
Tetris Render
VGA Controller
Display




# 4) Module Descriptions:

a. Module Descriptions:

1. Module: tetris.sv (containing tetris_core and tetris_render)
    - tetris_core:
        - Description: The tetris_core module is the central component of the Tetris game engine. It handles the core game mechanics, including piece movement, rotation, collision detection, and line clearing. This module maintains the state of the game board and the active piece.
        - Ports:
            - Input Ports: Reset, vsync, keycodes[31:0]
            - Output Ports: board[0:199][3:0], cur_shape[2:0], cur_orient[1:0], cur_row[4:0], cur_col[5:0], cur_clr[3:0], game_over
        - Purpose: This module serves as the brain of the Tetris game, processing player inputs and gravity to update the game state at a rate of approximately 3 Hz. It contains the logic for determining valid moves, locking pieces, clearing lines, and spawning new pieces.
    - tetris_render:
        - Description: The tetris_render module is responsible for converting the game state into visual output. It takes the current game board state, active piece information, and current pixel coordinates as inputs, and outputs the appropriate RGB values for each pixel.

        - Ports:
            - Input Ports: DrawX[9:0], DrawY[9:0], board[0:199][3:0], cur_shape[2:0], cur_orient[1:0], cur_row[4:0], cur_col[5:0], cur_clr[3:0], game_over
            - Output Ports: Red[3:0], Green[3:0], Blue[3:0]
        - Purpose: This module maps game board coordinates to pixel coordinates and determines the color of each pixel based on the game state. It handles rendering both the locked pieces on the game board and the active piece. When game_over is high, it renders the entire screen red.
2. Module: mb_usb_hdmi_top.sv
    - Description: This module serves as the top-level wrapper for the entire Tetris system, integrating the MicroBlaze processor, USB controller, and HDMI controller with the Tetris game modules.
    - Ports:
        - Input Ports: Clk, Reset_n, USB_DATA
        - Output Ports: USB_ADDR, hdmi_tmds_clk_n, hdmi_tmds_clk_p, hdmi_tmds_data_n, hdmi_tmds_data_p, hex_segA, hex_gridA, hex_segB, hex_gridB
    - Purpose: This module ties together all the components of the Tetris system, providing the interfaces to external devices like the keyboard and display. It also connects to the hex displays for score output.
3. Module: color_mapper.sv
    - Description: The color_mapper module determines the color of each pixel on the screen based on the game state information provided by the tetris modules.
    - Ports:
        - Input Ports: DrawX, DrawY, board_state, active_piece_info, game_over
        - Output Ports: Red, Green, Blue
    - Purpose: This module translates game state information into RGB color values for the display. It renders the game board, active piece, and game over screen.
4. Module: VGA_controller.sv
    - Description: The VGA_controller module generates the necessary signals for driving a VGA display. It manages the horizontal and vertical sync signals, generates pixel coordinates, and indicates when the display is in the active region.
    - Ports:
        - Input Ports: pixel_clk, reset
        - Output Ports: hs, vs, active_nblank, sync, drawX[9:0], drawY[9:0]
    - Purpose: This module provides the timing and coordinate generation necessary for display output. It generates the DrawX and DrawY signals that are used by the tetris_render module to determine which pixel to render.
5. Module: ball.sv (modified for Tetris)

    - Description: Originally designed for rendering a moving ball in Lab 6.2, this module has been extensively modified to handle Tetris piece rendering and movement.
    - Ports:
        - Input Ports: Reset, frame_clk, key_input[7:0]
        - Output Ports: piece_X[9:0], piece_Y[9:0], piece_shape[2:0], piece_orientation[1:0]
    - Purpose: This modified module controls the position and orientation of the active Tetris piece based on player input and game state.
6. Module: hex_driver.sv
    - Description: This module converts binary values into 7-segment display patterns for the FPGA's hexadecimal displays.
    - Ports:
        - Input Ports: in[3:0]
        - Output Ports: out[6:0]
    - Purpose: Used to display the player's score on the Urbana FPGA board's built-in 7-segment displays. Each hex display shows one digit of the score, allowing players to track their progress.

# 5) Design Statistics:

Tetris Final Project Design Statistics:

- LUT: 3245
- DSP: 0
- Memory(BRAM): 22
- Flip-Flop: 4283
- Latches: 0
- Frequency: 100.0 MHz
- Static Power: 0.082 W
- Dynamic Power: 0.422 W
- Total Power: 0.504 W

Our Tetris implementation makes efficient use of the Urbana FPGA board's resources while maintaining high performance. The design uses a moderate number of LUTs and flip-flops, with most of the logic dedicated to the core game mechanics and rendering. The use of BRAM for storing the game board and piece configurations allows for efficient memory utilization. The design operates at 100 MHz, providing ample processing power for the game logic and rendering, with a total power consumption of 0.504 W.

Compared to the Lab 6.2 from which we started, our Tetris implementation uses significantly more resources due to the additional game logic and rendering functionality. However, the increase in resource usage is justified by the enhanced functionality, with the Tetris implementation offering a complete gaming experience rather than just a simple ball animation.

The design achieves a good balance between resource utilization and performance, with sufficient processing power for smooth gameplay at 60 Hz while maintaining a moderate power consumption. The use of hardware acceleration for the core game logic and rendering allows for consistent frame rates and responsive gameplay, even as the game complexity increases.

# 6) Conclusion:

a. Functionality of the Design: Our Tetris implementation successfully achieved all the baseline functionality outlined in the project proposal. The game supports the full Tetris gameplay loop, including piece movement, rotation, line clearing, and score tracking. The hardware acceleration of the game engine ensures smooth gameplay at 60 Hz, with responsive input handling and consistent frame rates.

The design functioned as intended, with the hardware modules handling the core game mechanics and rendering, and the software components managing input processing and score tracking. The use of memory-mapped registers for communication between hardware and software components allowed for efficient data transfer and control.

Our implementation successfully displays the player's score on the Urbana FPGA board's hexadecimal displays, clears completed lines with appropriate scoring rewards, and shows a red screen when the game over condition is met. These features create a complete and engaging Tetris gaming experience.

b. Potential Extensions and Applications: There are several potential extensions to our Tetris implementation that could enhance the gameplay experience:

1. Increasing difficulty over time: The game could gradually increase the speed of piece descent as the player progresses, making the game more challenging over time.
2. Sound effects: Adding sound effects for piece movement, rotation, line clearing, and game over would enhance the gaming experience.
3. Next piece preview: Displaying the next piece to be spawned would allow players to plan their moves more strategically.
4. Hold piece functionality: Allowing players to hold a piece for later use would add an additional strategic element to the game.
5. More sophisticated scoring system: Implementing a scoring system that rewards combos and T-spins would add depth to the gameplay.

These extensions could be implemented by adding new hardware modules or enhancing the existing ones, and by updating the software components to support the new functionality.

c. Feedback on Lab Materials: The lab materials for Lab 6.2 provided a solid foundation for implementing the Tetris game. The example code, particularly the VGA controller and color mapper modules, was very helpful in getting started with the display aspect of the project. However, we had to significantly modify and extend these modules to support the more complex rendering requirements of Tetris.

One area where additional guidance would have been helpful is in the integration of game logic with display rendering. More detailed information on how to efficiently implement game mechanics in hardware would have made this aspect of the project easier to implement.

Overall, the lab materials provided a good starting point for the project, allowing us to focus on the implementation details while having a clear understanding of the basic Urbana FPGA-based display system. Our success in transforming the simple ball-based animation from Lab 6.2 into a fully functional Tetris game demonstrates the flexibility of the Urbana FPGA platform and the effectiveness of the lab materials in providing a strong foundation for more complex projects.